

# CoCon: A Confidentiality-Verified Conference Management System

Andrei Popescu      Peter Lammich      Thomas Bauereiss

February 6, 2026

## Abstract

This entry contains the confidentiality verification of the (functional kernel of) the CoCon conference management system [3, 6]. The confidentiality properties refer to the documents managed by the system, namely papers, reviews, discussion logs and acceptance/rejection decisions, and also to the assignment of reviewers to papers. They have all been formulated as instances of BD Security [4, 5] and verified using the BD Security unwinding technique.

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	The basic types . . . . .	5
1.2	Conference, user and paper IDs . . . . .	7
<b>2</b>	<b>System specification</b>	<b>9</b>
2.1	System state . . . . .	9
2.2	The actions . . . . .	12
2.2.1	Initialization of the system . . . . .	12
2.2.2	Actions unbound by any existing conference (with its phases) . . . . .	13
2.2.3	Actions available in the noPH phase . . . . .	16
2.2.4	Actions available in the setPH phase . . . . .	16
2.2.5	Actions available starting from the setPH phase . . . . .	17
2.2.6	Actions available in the subPH phase . . . . .	18
2.2.7	Actions available starting from the subPH phase . . . . .	20
2.2.8	Actions available in the bidPH phase . . . . .	21
2.2.9	Actions available starting from the bidPH phase . . . . .	22
2.2.10	Actions available in the revPH phase . . . . .	22
2.2.11	Actions available starting from the revPH phase . . . . .	23
2.2.12	Actions available in the disPH phase . . . . .	24
2.2.13	Actions available starting from the disPH phase . . . . .	25

2.2.14	Actions available starting from the notifPH phase . . .	26
2.3	The step function . . . . .	27
<b>3</b>	<b>Safety properties</b>	<b>41</b>
3.1	Infrastructure for invariance reasoning . . . . .	41
3.2	Safety proofs . . . . .	44
3.3	Properties of the step function . . . . .	61
3.4	Action-safety properties . . . . .	63
3.5	Miscellaneous . . . . .	64
<b>4</b>	<b>Observation setup for confidentiality properties</b>	<b>70</b>
<b>5</b>	<b>Paper Confidentiality</b>	<b>70</b>
5.1	Preliminaries . . . . .	71
5.2	Value Setup . . . . .	75
5.3	Confidentiality protection from users who are not the paper's authors or PC members . . . . .	77
5.4	Confidentiality protection from non-authors . . . . .	86
<b>6</b>	<b>Review Confidentiality</b>	<b>95</b>
6.1	Preliminaries . . . . .	96
6.2	Value Setup . . . . .	107
6.3	Confidentiality protection from users who are not the review's author . . . . .	110
6.4	Confidentiality protection from users who are not the review's author or a PC member . . . . .	125
6.5	Confidentiality from users who are not the review's author, a PC member, or an author of the paper . . . . .	142
<b>7</b>	<b>Discussion Confidentiality</b>	<b>154</b>
7.1	Preliminaries . . . . .	154
7.2	Value Setup . . . . .	158
7.3	Confidentiality protection from non-PC-members . . . . .	160
<b>8</b>	<b>Decision Confidentiality</b>	<b>169</b>
8.1	Preliminaries . . . . .	170
8.2	Value Setup . . . . .	178
8.3	Confidentiality protection from non-PC-members . . . . .	180
8.4	Confidentiality protection from users who are not PC mem- bers or authors of the paper . . . . .	193
<b>9</b>	<b>Reviewer Assignment Confidentiality</b>	<b>204</b>
9.1	Preliminaries . . . . .	205
9.2	Value Setup . . . . .	215
9.3	Confidentiality protection from non-PC-members . . . . .	220

9.4 Confidentiality protection from users who are not PC members or authors of the paper . . . . .	246
<b>10 Traceback properties</b>	<b>262</b>
10.1 Preliminaries . . . . .	263
10.2 Authorship . . . . .	264
10.3 Becoming a Conference Chair . . . . .	266
10.4 Committee Membership . . . . .	268
10.5 Being a Reviewer . . . . .	270
10.6 Conflicts . . . . .	271
10.7 Conference Phases . . . . .	274

## 1 Introduction

This document presents the confidentiality verification of the (functional kernel of) the CoCon conference management system [3, 6]. CoCon was the first case study of BD Security, a general framework for the specification and verification of information flow security. The framework works for any input/output (I/O) automata and allows the specification of flexible policies for information flow security by describing the observations, the secrets, a bound on information release (also known as “declassification bound”) and a trigger for information release (also known as “declassification trigger”).

In CoCon, a conference goes through several phases:

**No-Phase** Any user can apply for a new conference, with the effect of registering it in the system as initially having “no phase.” After approval from CoCon’s superuser,<sup>1</sup> the conference moves to the setup phase, with the applicant becoming a conference chair.

**Setup** A conference chair can add new chairs and new regular PC members. From here on, advancing the conference through its different phases can be done by the chairs.

**Submission** Any user can list the conferences awaiting submissions (i.e., being in the submission phase). A user can submit new papers, upload new versions of their existing papers, or indicate other users as coauthors thereby granting them reading and editing rights.

**Bidding** Authors are no longer allowed to upload or register new papers, and PC members are allowed to view the submitted papers. PC members can place bids, indicating for each paper one of the following preferences: “want to review”, “would review”, “no preference”, “would not review”, and “conflict”. If the preference is “conflict”, the PC

---

<sup>1</sup>The superuser’s powers are restricted to approving or rejecting new conference requests.

member cannot be assigned that paper, and will not see its discussion. “Conflict” is assigned automatically to papers authored by a PC member.

**Reviewing** Chairs can assign reviewers to papers, which must be among the PC members who have no conflict with given paper. The assigned reviewers can edit their reviews.

**Discussion** All PC members having no conflict with a paper can see its reviews and can add comments. The reviewers can still edit their reviews, but in a transparent manner—so that the overwritten versions are still visible to the non-conflict PC members. Also, chairs can edit the decision.

**Notification** The authors can read the reviews and the accept/reject decision, which no one can edit any longer.

After this introduction and a section on technical preliminaries, this document presents the specification of the CoCon system, as an input/output (I/O) automaton. Following is a section on proved safety properties about the system (invariants) that are needed in the proofs of confidentiality.

The confidentiality properties of CoCon are formalized as instances of BD Security. They cover confidentiality aspects about:

- papers
- reviews of papers
- discussion logs consisting of comments from the PC members
- decisions on the papers’ acceptance or rejection
- assignment of reviewers to papers

Each of these types of confidentiality properties have dedicated sections (and corresponding folders in the formalization) with self-explanatory names. BD Security is defined in terms of an observation infrastructure, a secrecy infrastructure, a declassification trigger and a declassification bound. The observations are always given by an arbitrary set of users (which is fixed in the “Observation Setup” section). The secrets (called “values” in this formalization) and the declassification bounds and triggers are specific to each property. The bounds and triggers are chosen in such a way that their interplay covers the entire spectrum of information flow through the system in relation to the given secrets. This is explained in [6, Section 3.5].

The proofs proceed using the method of BD Security unwinding, which is part of the AFP entry on BD Security [5] and is described in detail in [6, Sections 4.1 and 4.2] and [4, Sections 2.5 and 2.6]. For managing proof

complexity, we take a modular approach, building several unwinding relations that are connected in a sequence and have an exit point into an error component. This approach is presented in [6] as Corollary 6 (Sequential Unwinding Theorem) and in [4] as Theorem 4 (Sequential Multiplex Unwinding Theorem).

The last section formalizes what we call *traceback properties*,<sup>2</sup> which strengthen the confidentiality guarantees. Confidentiality formulated as BD security states properties of essentially the following form: “Unless a user acquires such and such role and/or the conference reaches such and such phase, that user cannot learn such and such information.” Traceback properties show that it is not possible for a user to usurp such roles, and that the conference only progresses through different phases in a “legal” way. [6, Section 3.6] explains CoCon’s traceback properties in detail.

As a matter of notation, this formalization (similarly to all our AFP formalizations involving BD security) concurs with the original conference paper on CoCon [3] and differs from the later journal paper [6] in that the secrets are called “values” (and consequently the type of secrets is denoted by “value”), and are ranged over by “v” rather than “s”. On the other hand, we use “s” (rather than “ $\sigma$ ”) to range over states. Moreover, the formalization uses the following notations for the various BD security components:

- phi for the secret discriminator isSec
- f for the secret selector getSec
- gamma for the observation discriminator isObs
- g for the observation selector getObs

**theory** *Prelim*

**imports** *Fresh-Identifiers.Fresh-String Bounded-Deducibility-Security.Trivia*  
**begin**

## 1.1 The basic types

**type-synonym** *string* = *String.literal*

**definition** *emptyStr* = *STR ""*

**type-synonym** *phase* = *nat*

**abbreviation** *noPH*  $\equiv$  *(0::nat)*    **abbreviation** *setPH*  $\equiv$  *Suc noPH*    **abbrevi-**

**ation** *subPH*  $\equiv$  *Suc setPH*

**abbreviation** *bidPH*  $\equiv$  *Suc subPH*    **abbreviation** *revPH*  $\equiv$  *Suc bidPH*    **abbrevi-**

**ation** *disPH*  $\equiv$  *Suc revPH*

---

<sup>2</sup>In previous work, we called these types of properties *accountability properties* [1, 2] or *forensic properties* [3]. The *traceback properties* terminology is used in [6].

**abbreviation** *notifPH*  $\equiv$  *Suc disPH* **abbreviation** *closedPH*  $\equiv$  *Suc notifPH*

**fun** *SucPH* **where**  
*SucPH ph* = (if *ph* = *closedPH* then *closedPH* else *Suc ph*)

**datatype** *user* = *User string string string*  
**fun** *nameUser* **where** *nameUser (User name info email)* = *name*  
**fun** *infoUser* **where** *infoUser (User name info email)* = *info*  
**fun** *emailUser* **where** *emailUser (User name info email)* = *email*  
**definition** *emptyUser*  $\equiv$  *User emptyStr emptyStr emptyStr*

**typedecl** *raw-data*  
**code-printing type-constructor** *raw-data*  $\rightarrow$  (*Scala*) *java.io.File*

**datatype** *pcontent* = *NoPContent* | *PContent raw-data*

**datatype** *score* = *NoScore* | *MinusThree* | *MinusTwo* | *MinusOne* | *Zero* | *One* | *Two* | *Three*

**fun** *scoreAsInt* :: *score*  $\Rightarrow$  *int* **where**  
*scoreAsInt MinusThree* = -3  
*scoreAsInt MinusTwo* = -2  
*scoreAsInt MinusOne* = -1  
*scoreAsInt Zero* = 0  
*scoreAsInt One* = 1  
*scoreAsInt Two* = 2  
*scoreAsInt Three* = 3

**datatype** *exp* = *NoExp* | *Zero* | *One* | *Two* | *Three* | *Four*

**type-synonym** *rcontent* = *exp \* score \* string*  
**fun** *scoreOf* :: *rcontent*  $\Rightarrow$  *score* **where** *scoreOf (exp,sc,txt)* = *sc*

**type-synonym** *review* = *rcontent list*

**abbreviation** *emptyReview* :: *review* **where** *emptyReview*  $\equiv$  []  
**datatype** *discussion* = *Dis string list*  
**definition** *emptyDis*  $\equiv$  *Dis* []  
**datatype** *decision* = *NoDecision* | *Accept* | *Reject*

**datatype** *paper* = *Paper string string pcontent review list discussion decision list*

**fun** *titlePaper* **where** *titlePaper (Paper title abstract content reviews dis decs)* = *title*

```

fun abstractPaper where abstractPaper (Paper title abstract content reviews dis
decs) = abstract
fun contentPaper where contentPaper (Paper title abstract content reviews dis
decs) = content
fun reviewsPaper where reviewsPaper (Paper title abstract content reviews dis
decs) = reviews
fun disPaper where disPaper (Paper title abstract content reviews dis decs) = dis

fun decsPaper where decsPaper (Paper title abstract content reviews dis decs) =
decs

fun decPaper where decPaper pap = hd (decsPaper pap)

```

```

definition emptyPaper :: paper where
emptyPaper ≡ Paper emptyStr emptyStr NoPContent [] emptyDis []

```

```

datatype conf = Conf string string
fun nameConf where nameConf (Conf name info) = name
fun infoConf where infoConf (Conf name info) = info
definition emptyConf ≡ Conf emptyStr emptyStr

```

```

datatype password = Password string
definition emptyPass ≡ Password emptyStr

```

```

datatype preference = NoPref | Want | Would | WouldNot | Conflict

```

## 1.2 Conference, user and paper IDs

```

datatype userID = UserID string
datatype paperID = PaperID string
datatype confID = ConfID string

```

```

definition emptyUserID ≡ UserID emptyStr
definition voronkovUserID ≡ UserID (STR "voronkov")
definition emptyPaperID ≡ PaperID emptyStr
definition emptyConfID ≡ ConfID emptyStr

```

```

datatype role = Aut paperID | Rev paperID nat | PC | Chair
fun isRevRole where isRevRole (Rev - -) = True | isRevRole - = False

```

```

fun isRevRoleFor :: paperID ⇒ role ⇒ bool where
  isRevRoleFor papID (Rev papID' n) ↔ papID = papID'
| isRevRoleFor papID - ↔ False

```

```

fun userIDAsStr where userIDAsStr (UserID str) = str

definition getFreshUserID userIDs  $\equiv$  UserID (fresh (set (map userIDAsStr userIDs))
  (STR "1"))

lemma UserID-userIDAsStr[simp]: UserID (userIDAsStr userID) = userID
by (cases userID) auto

lemma member-userIDAsStr-iff[simp]: str  $\in$  userIDAsStr ‘ (set userIDs)  $\longleftrightarrow$ 
  UserID str  $\in\in$  userIDs
by (metis UserID-userIDAsStr image-iff userIDAsStr.simps)

lemma getFreshUserID:  $\neg$  getFreshUserID userIDs  $\in\in$  userIDs
  using fresh-notIn[of set (map userIDAsStr userIDs) STR "1"]
  by (auto simp: getFreshUserID-def)

instantiation UserID :: linorder
begin
definition le-userID-def: uid  $\leq$  uid'  $\equiv$  case (uid,uid') of (UserID str, UserID
  str')  $\Rightarrow$  str  $\leq$  str'
definition lt-userID-def: uid < uid'  $\equiv$  case (uid,uid') of (UserID str, UserID
  str')  $\Rightarrow$  str < str'
instance by standard (auto simp: le-userID-def lt-userID-def split: userID.splits)
end

fun paperIDAsStr where paperIDAsStr (PaperID str) = str

definition getFreshPaperID paperIDs  $\equiv$  PaperID (fresh (set (map paperIDAsStr
  paperIDs)) (STR "2"))

lemma PaperID-paperIDAsStr[simp]: PaperID (paperIDAsStr paperID) = paperID
by (cases paperID) auto

lemma member-paperIDAsStr-iff[simp]: str  $\in$  paperIDAsStr ‘ paperIDs  $\longleftrightarrow$  Pa-
  perID str  $\in$  paperIDs
by (metis PaperID-paperIDAsStr image-iff paperIDAsStr.simps)

lemma getFreshPaperID:  $\neg$  getFreshPaperID paperIDs  $\in\in$  paperIDs
  using fresh-notIn[of set (map paperIDAsStr paperIDs) STR "2"]
  by (auto simp: getFreshPaperID-def)

instantiation PaperID :: linorder
begin
definition le-paperID-def: uid  $\leq$  uid'  $\equiv$  case (uid,uid') of (PaperID str, PaperID
  str')  $\Rightarrow$  str  $\leq$  str'
definition lt-paperID-def: uid < uid'  $\equiv$  case (uid,uid') of (PaperID str, PaperID
  str')  $\Rightarrow$  str < str'
instance by standard (auto simp: le-paperID-def lt-paperID-def split: paperID.splits)

```

**end**

**fun** *confIDAsStr* **where** *confIDAsStr* (*ConfID str*) = *str*

**definition** *getFreshConfID confIDs*  $\equiv$  *ConfID* (*fresh* (*set* (*map confIDAsStr confIDs*))) (*STR "2"*)

**lemma** *ConfID-confIDAsStr[simp]*: *ConfID* (*confIDAsStr confID*) = *confID*  
**by** (*cases confID*) *auto*

**lemma** *member-confIDAsStr-iff[simp]*: *str*  $\in$  *confIDAsStr* ‘(*set confIDs*)  $\longleftrightarrow$  *ConfID str*  $\in$  *confIDs*  
**by** (*metis ConfID-confIDAsStr image-iff confIDAsStr.simps*)

**lemma** *getFreshConfID*:  $\neg$  *getFreshConfID confIDs*  $\in$  *confIDs*  
**using** *fresh-notIn*[*of set (map confIDAsStr confIDs) STR "2"*]  
**by** (*auto simp: getFreshConfID-def*)

**instantiation** *confID* :: *linorder*

**begin**

**definition** *le-confID-def*: *uid*  $\leq$  *uid'*  $\equiv$  *case* (*uid,uid'*) *of* (*ConfID str, ConfID str'*)  $\Rightarrow$  *str*  $\leq$  *str'*

**definition** *lt-confID-def*: *uid*  $<$  *uid'*  $\equiv$  *case* (*uid,uid'*) *of* (*ConfID str, ConfID str'*)  $\Rightarrow$  *str*  $<$  *str'*

**instance** **by** *standard* (*auto simp: le-confID-def lt-confID-def split: confID.splits*)  
**end**

**end**

## 2 System specification

This section formalizes the CoCon system as an I/O automaton. We call the inputs “actions”.

**theory** *System-Specification*

**imports** *Prelim*

**begin**

### 2.1 System state

The superuser of the system is called “voronkov”, as a form acknowledgement for our inspiration from EasyChair when creating CoCon.

**record** *state* =

*confIDs* :: *confID list*

*conf* :: *confID*  $\Rightarrow$  *conf*

*userIDs* :: *userID* list  
*pass* :: *userID* ⇒ *password*  
*user* :: *userID* ⇒ *user*  
*roles* :: *confID* ⇒ *userID* ⇒ *role* list

*paperIDs* :: *confID* ⇒ *paperID* list  
*paper* :: *paperID* ⇒ *paper*

*pref* :: *userID* ⇒ *paperID* ⇒ *preference*

*voronkov* :: *userID*

*news* :: *confID* ⇒ *string* list  
*phase* :: *confID* ⇒ *phase*

**abbreviation** *isPC* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *bool* **where**  
*isPC* *s confID userID* ≡ *PC* ∈∈ *roles s confID userID*  
**abbreviation** *isChair* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *bool* **where**  
*isChair* *s confID userID* ≡ *Chair* ∈∈ *roles s confID userID*  
**abbreviation** *isAut* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *paperID* ⇒ *bool* **where**  
*isAut* *s confID userID papID* ≡ *Aut papID* ∈∈ *roles s confID userID*  
**definition** *isAutSome* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *bool* **where**  
*isAutSome* *s confID userID* ≡ *list-ex* (*isAut* *s confID userID*) (*paperIDs* *s confID*)

**definition** *authors* :: *state* ⇒ *confID* ⇒ *paperID* ⇒ *userID* list **where**  
*authors* *s confID papID* ≡ *filter* ( $\lambda$  *userID*. *isAut* *s confID userID papID*) (*userIDs* *s*)  
**abbreviation** *isRevNth* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *paperID* ⇒ *nat* ⇒ *bool*  
**where**

*isRevNth* *s confID userID papID n* ≡ *Rev papID n* ∈∈ *roles s confID userID*  
**definition** *isRev* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *paperID* ⇒ *bool* **where**  
*isRev* *s confID userID papID* ≡ *list-ex* (*isRevRoleFor* *papID*) (*roles s confID userID*)

**definition** *getRevRole* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *paperID* ⇒ *role* option  
**where**  
*getRevRole* *s confID userID papID* ≡ *List.find* (*isRevRoleFor* *papID*) (*roles s confID userID*)

**definition** *getNthReview* :: *state* ⇒ *paperID* ⇒ *nat* ⇒ *review* **where**  
*getNthReview* *s papID n* ≡ (*reviewsPaper* (*paper s papID*))!n

**definition** *getAllPaperIDs* :: *state* ⇒ *paperID* list **where**  
*getAllPaperIDs* *s* ≡ *concat* [*paperIDs* *s confID*. *confID* ← *confIDs* *s*]  
**definition** *getReviewIndex* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *paperID* ⇒ *nat* **where**

*getReviewIndex s confID uID papID*  $\equiv$   
*case getRevRole s confID uID papID of Some (Rev papID' n)  $\Rightarrow$  n*

**definition** *getReviewersReviews* :: *state*  $\Rightarrow$  *confID*  $\Rightarrow$  *paperID*  $\Rightarrow$  (*userID* \* *review*)  
*list where*  
*getReviewersReviews s confID papID*  $\equiv$   
 [(*uID*, *getNthReview s papID (getReviewIndex s confID uID papID)*),  
   *uID*  $\leftarrow$  *userIDs s*,  
   *isRev s confID uID papID*  
 ]

**definition** *isAUT* :: *state*  $\Rightarrow$  *userID*  $\Rightarrow$  *paperID*  $\Rightarrow$  *bool where*  
*isAUT s uID papID*  $\equiv$   $\exists$  *confID*. *isAut s confID uID papID*

**definition** *isREVNth* :: *state*  $\Rightarrow$  *userID*  $\Rightarrow$  *paperID*  $\Rightarrow$  *nat*  $\Rightarrow$  *bool where*  
*isREVNth s uID papID n*  $\equiv$   $\exists$  *confID*. *isRevNth s confID uID papID n*

**lemma** *isRev-getRevRole*:  
**assumes** *isRev s confID uID papID*  
**shows** *getRevRole s confID uID papID  $\neq$  None*  
**using** *assms list-ex-find unfolding isRev-def getRevRole-def by auto*

**lemma** *getRevRole-Some*:  
**assumes** *getRevRole s confID uID papID = Some role*  
**shows**  $\exists$  *n*. *role = Rev papID n*  
**using** *assms unfolding getRevRole-def unfolding find-Some-iff apply (cases role, auto)*  
**by** (*metis isRevRoleFor.simps*) $+$

**lemma** *isRev-getRevRole2*:  
**assumes** *isRev s confID uID papID***shows**  $\exists$  *n*. *getRevRole s confID uID papID = Some (Rev papID n)*  
**using** *assms getRevRole-Some by (cases getRevRole s confID uID papID) (auto simp: isRev-getRevRole)*

**lemma** *isRev-imp-isRevNth-getReviewIndex*:  
**assumes** *isRev s confID uID papID*  
**shows** *isRevNth s confID uID papID (getReviewIndex s confID uID papID)*  
**proof** –  
  **obtain** *n where 1: getRevRole s confID uID papID = Some (Rev papID n)*  
  **using** *isRev-getRevRole2[OF assms] by auto*  
  **hence** *isRevNth s confID uID papID n*  
  **unfolding** *getRevRole-def unfolding find-Some-iff by auto*  
  **moreover have** *getReviewIndex s confID uID papID = n* **using** *1 unfolding getReviewIndex-def by simp*  
  **ultimately show** *?thesis* **by auto**  
**qed**

**lemma** *isRev-def2*:  
*isRev s confID uID papID*  $\longleftrightarrow$   $(\exists n. \text{isRevNth } s \text{ confID } uID \text{ papID } n)$  (**is** ?A  $\longleftrightarrow$  ?B)  
**proof**  
  **assume** ?A **thus** ?B **using** *isRev-imp-isRevNth-getReviewIndex* **by** *blast*  
**next**  
  **assume** ?B **thus** ?A **unfolding** *isRev-def list-ex-iff* **by** *force*  
**qed**

**lemma** *isRev-def3*:  
*isRev s confID uID papID*  $\longleftrightarrow$  *isRevNth s confID uID papID (getReviewIndex s confID uID papID)*  
**by** (*metis isRev-def2 isRev-imp-isRevNth-getReviewIndex*)

**lemma** *getFreshPaperID-getAllPaperIDs[simp]*:  
  **assumes** *confID*  $\in \in$  *confIDs s*  
  **shows**  $\neg$  *getFreshPaperID (getAllPaperIDs s)  $\in \in$  paperIDs s confID*  
  **using** *assms getFreshPaperID[of getAllPaperIDs s]*  
  **by** (*auto simp: getAllPaperIDs-def*)

**lemma** *getRevRole-Some-Rev*:  
*getRevRole s cid uid pid = Some (Rev pid' n)*  $\implies$  *pid' = pid*  
**by** (*metis getRevRole-Some role.inject*)

**lemma** *getRevRole-Some-Rev-isRevNth*:  
*getRevRole s cid uid pid = Some (Rev pid' n)*  $\implies$  *isRevNth s cid uid pid n*  
  **unfolding** *getRevRole-def find-Some-iff*  
  **apply** (*elim exE*)  
  **subgoal for** *i*  
    **apply**(*cases roles s cid uid ! i*)  
    **apply** *auto*  
    **by** (*metis nth-mem*)  
  **done**

**definition** *IDsOK* :: *state*  $\Rightarrow$  *confID list*  $\Rightarrow$  *userID list*  $\Rightarrow$  *paperID list*  $\Rightarrow$  *bool*  
**where**  
*IDsOK s cIDs uIDs papIDs*  $\equiv$   
  *list-all*  $(\lambda \text{confID}. \text{confID} \in \in \text{confIDs } s)$  *cIDs*  $\wedge$   
  *list-all*  $(\lambda \text{uID}. \text{uID} \in \in \text{userIDs } s)$  *uIDs*  $\wedge$   
  *list-all*  $(\lambda \text{papID}. \text{papID} \in \in \text{paperIDs } s (\text{hd } \text{cIDs}))$  *papIDs*

## 2.2 The actions

### 2.2.1 Initialization of the system

**definition** *istate* :: *state*  
**where**

```

istate ≡
(
  confIDs = [],
  conf = (λ confID. emptyConf),
  userIDs = [voronkovUserID],
  pass = (λ uID. emptyPass),
  user = (λ uID. emptyUser),
  roles = (λ confID uID. []),
  paperIDs = (λ confID. []),
  paper = (λ papID. emptyPaper),
  pref = (λ uID papID. NoPref),
  voronkov = voronkovUserID,
  news = (λ confID. []),
  phase = (λ confID. noPH)
)

```

### 2.2.2 Actions unbound by any existing conference (with its phases)

**definition**  $createUser :: state \Rightarrow userID \Rightarrow password \Rightarrow string \Rightarrow string \Rightarrow string \Rightarrow state$

**where**

```

createUser s uID p name info email ≡
  let uIDs = userIDs s
  in
  s (|userIDs := uID # uIDs,
     user := (user s) (uID := User name info email),
     pass := (pass s) (uID := p)|)

```

**definition**  $e\text{-createUser} :: state \Rightarrow userID \Rightarrow password \Rightarrow string \Rightarrow string \Rightarrow string \Rightarrow bool$  **where**

```

e-createUser s uID p name info email ≡
  ¬ uID ∈∈ userIDs s

```

**definition**  $updateUser :: state \Rightarrow userID \Rightarrow password \Rightarrow password \Rightarrow string \Rightarrow string \Rightarrow string \Rightarrow state$

**where**

```

updateUser s uID p p' name info email ≡
  s (|user := (user s) (uID := User name info email),
     pass := (pass s) (uID := p')|)

```

**definition**  $e\text{-updateUser} :: state \Rightarrow userID \Rightarrow password \Rightarrow password \Rightarrow string \Rightarrow string \Rightarrow string \Rightarrow bool$

**where**

```

e-updateUser s uID p p' name info email ≡
  IDsOK s [] [uID] [] ∧ pass s uID = p

```

**definition**  $readAmIVoronkov :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$

**where**

$readAmIVoronkov\ s\ uID\ p \equiv$   
 $uID = voronkov\ s$

**definition**  $e-readAmIVoronkov :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$

**where**

$e-readAmIVoronkov\ s\ uID\ p \equiv$   
 $IDsOK\ s\ []\ [uID]\ [] \wedge pass\ s\ uID = p$

**definition**  $readUser :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow string * string * string$

**where**

$readUser\ s\ uID\ p\ uID' \equiv$   
 $case\ user\ s\ uID'\ of\ User\ name\ info\ email \Rightarrow (name, info, email)$

**definition**  $e-readUser :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow bool$

**where**

$e-readUser\ s\ uID\ p\ uID' \equiv$   
 $IDsOK\ s\ []\ [uID, uID']\ [] \wedge pass\ s\ uID = p$

**definition**  $createConf :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow string \Rightarrow string \Rightarrow state$

**where**

$createConf\ s\ confID\ uID\ p\ name\ info \equiv$   
 $let\ confIDs = confIDs\ s$   
 $in$   
 $s\ (\downarrow confIDs := confID\ \# confIDs,$   
 $conf := (conf\ s)\ (confID := Conf\ name\ info),$   
 $roles := fun-upd2\ (roles\ s)\ confID\ uID\ [PC, Chair]$   
 $\downarrow)$

**definition**  $e-createConf :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow string \Rightarrow string \Rightarrow bool$

**where**

$e-createConf\ s\ confID\ uID\ p\ name\ info \equiv$   
 $IDsOK\ s\ []\ [uID]\ [] \wedge pass\ s\ uID = p \wedge$   
 $\neg confID \in \in confIDs\ s$

**definition**  $readConf :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow string * string * (role\ list) * phase$

**where**

$readConf\ s\ confID\ uID\ p \equiv$   
 $(nameConf\ (conf\ s\ confID),\ infoConf\ (conf\ s\ confID),$   
 $[rl \leftarrow roles\ s\ confID\ uID.\ \neg\ isRevRole\ rl],\ phase\ s\ confID)$

**definition**  $e-readConf :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow bool$

**where**

$e-readConf\ s\ confID\ uID\ p \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [] \wedge\ pass\ s\ uID = p$

**definition**  $listConfs :: state \Rightarrow userID \Rightarrow password \Rightarrow confID\ list$

**where**

$listConfs\ s\ uID\ p \equiv$   
 $confIDs\ s$

**definition**  $e-listConfs :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$

**where**

$e-listConfs\ s\ uID\ p \equiv$   
 $IDsOK\ s\ []\ [uID]\ [] \wedge\ pass\ s\ uID = p \wedge$   
 $uID = voronkov\ s$

**definition**  $listAConfs :: state \Rightarrow userID \Rightarrow password \Rightarrow confID\ list$

**where**

$listAConfs\ s\ uID\ p \equiv$   
 $[confID.\ confID \leftarrow confIDs\ s,\ phase\ s\ confID = noPH]$

**definition**  $e-listAConfs :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$

**where**

$e-listAConfs\ s\ uID\ p \equiv$   
 $IDsOK\ s\ []\ [uID]\ [] \wedge\ pass\ s\ uID = p \wedge$   
 $uID = voronkov\ s$

**definition**  $listSConfs :: state \Rightarrow userID \Rightarrow password \Rightarrow confID\ list$

**where**

$listSConfs\ s\ uID\ p \equiv$   
 $[confID.\ confID \leftarrow confIDs\ s,\ phase\ s\ confID = subPH]$

**definition**  $e-listSConfs :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$

**where**

$e-listSConfs\ s\ uID\ p \equiv$   
 $IDsOK\ s\ []\ [uID]\ [] \wedge\ pass\ s\ uID = p$

**definition**  $listMyConfs :: state \Rightarrow userID \Rightarrow password \Rightarrow confID\ list$

**where**

$listMyConfs\ s\ uID\ p \equiv$

$[confID. confID \leftarrow confIDs\ s, roles\ s\ confID\ uID \neq []]$

**definition**  $e-listMyConfs :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$

**where**

$e-listMyConfs\ s\ uID\ p \equiv$   
 $IDsOK\ s\ []\ [uID]\ [] \wedge pass\ s\ uID = p$

**definition**  $listAllUsers :: state \Rightarrow userID \Rightarrow password \Rightarrow userID\ list$

**where**

$listAllUsers\ s\ uID\ p \equiv$   
 $userIDs\ s$

**definition**  $e-listAllUsers :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$

**where**

$e-listAllUsers\ s\ uID\ p \equiv$   
 $IDsOK\ s\ []\ [uID]\ [] \wedge pass\ s\ uID = p$

**definition**  $listAllPapers :: state \Rightarrow userID \Rightarrow password \Rightarrow paperID\ list$

**where**

$listAllPapers\ s\ uID\ p \equiv$   
 $getAllPaperIDs\ s$

**definition**  $e-listAllPapers :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$

**where**

$e-listAllPapers\ s\ uID\ p \equiv$   
 $IDsOK\ s\ []\ [uID]\ [] \wedge pass\ s\ uID = p$

### 2.2.3 Actions available in the noPH phase

**definition**  $updateConfA :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow state$

**where**

$updateConfA\ s\ confID\ uID\ p \equiv$   
 $s\ (\backslash phase := (phase\ s)\ (confID := setPH))$

**definition**  $e-updateConfA :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow bool$

**where**

$e-updateConfA\ s\ confID\ uID\ p \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [] \wedge pass\ s\ uID = p \wedge$   
 $uID = voronkov\ s \wedge phase\ s\ confID = noPH$

### 2.2.4 Actions available in the setPH phase

**definition**  $createPC :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow state$

**where**

$createPC\ s\ confID\ uID\ p\ uID' \equiv$   
 $let\ rls = roles\ s\ confID\ uID'$   
 $in$   
 $s\ (\backslash roles := fun-upd2\ (roles\ s)\ confID\ uID'\ (List.insert\ PC\ rls))$

**definition**  $e\text{-createPC} :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow bool$   
**where**  
 $e\text{-createPC } s \text{ confID } uID \text{ p } uID' \equiv$   
 $\text{let } uIDs = \text{userIDs } s$   
 $\text{in}$   
 $IDsOK \ s \ [confID] \ [uID, uID'] \ [] \wedge \text{pass } s \ uID = p \wedge$   
 $\text{phase } s \ confID = \text{setPH} \wedge \text{isChair } s \ confID \ uID$

**definition**  $\text{createChair} :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow state$   
**where**  
 $\text{createChair } s \ confID \ uID \ p \ uID' \equiv$   
 $\text{let } rls = \text{roles } s \ confID \ uID'$   
 $\text{in}$   
 $s \ (\text{roles} := \text{fun-upd2 } (\text{roles } s) \ confID \ uID' \ (\text{List.insert } PC \ (\text{List.insert } Chair \ rls)))$

**definition**  $e\text{-createChair} :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow bool$   
**where**  
 $e\text{-createChair } s \ confID \ uID \ p \ uID' \equiv$   
 $\text{let } uIDs = \text{userIDs } s$   
 $\text{in}$   
 $IDsOK \ s \ [confID] \ [uID, uID'] \ [] \wedge \text{pass } s \ uID = p \wedge$   
 $\text{phase } s \ confID = \text{setPH} \wedge \text{isChair } s \ confID \ uID$

### 2.2.5 Actions available starting from the setPH phase

**definition**  $\text{updatePhase} :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow phase \Rightarrow state$   
**where**  
 $\text{updatePhase } s \ confID \ uID \ p \ ph \equiv$   
 $s \ (\text{phase} := (\text{phase } s) \ (\text{confID} := \text{ph}))$

**definition**  $e\text{-updatePhase} :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow phase \Rightarrow bool$  **where**  
 $e\text{-updatePhase } s \ confID \ uID \ p \ ph \equiv$   
 $IDsOK \ s \ [confID] \ [uID] \ [] \wedge \text{pass } s \ uID = p \wedge$   
 $\text{phase } s \ confID \geq \text{setPH} \wedge \text{phase } s \ confID < \text{closedPH} \wedge \text{isChair } s \ confID \ uID \wedge$   
 $\text{ph} = \text{SucPH } (\text{phase } s \ confID)$

**definition**  $\text{uupdateNews} :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow string \Rightarrow state$   
**where**  
 $\text{uupdateNews } s \ confID \ uID \ p \ ev \equiv$   
 $\text{let } evs = \text{news } s \ confID$

*in*  
 $s \langle news := (news\ s) (confID := ev \# evs) \rangle$

**definition**  $e\text{-updateNews} :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow string \Rightarrow bool$

**where**

$e\text{-updateNews}\ s\ confID\ uID\ p\ ev \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [] \wedge pass\ s\ uID = p \wedge$   
 $phase\ s\ confID \geq setPH \wedge phase\ s\ confID < closedPH \wedge isChair\ s\ confID\ uID$

**definition**  $readNews :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow string\ list$

**where**

$readNews\ s\ confID\ uID\ p \equiv$   
 $news\ s\ confID$

**definition**  $e\text{-readNews} :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow bool$

**where**

$e\text{-readNews}\ s\ confID\ uID\ p \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [] \wedge pass\ s\ uID = p \wedge$   
 $phase\ s\ confID \geq setPH \wedge isPC\ s\ confID\ uID$

**definition**  $listPC :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow userID\ list$

**where**

$listPC\ s\ confID\ uID\ p \equiv$   
 $[uID.\ uID \leftarrow userIDs\ s,\ isPC\ s\ confID\ uID]$

**definition**  $e\text{-listPC} :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow bool$

**where**

$e\text{-listPC}\ s\ confID\ uID\ p \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [] \wedge pass\ s\ uID = p \wedge$   
 $(phase\ s\ confID \geq subPH \vee (phase\ s\ confID \geq setPH \wedge isChair\ s\ confID\ uID))$

**definition**  $listChair :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow userID\ list$

**where**

$listChair\ s\ confID\ uID\ p \equiv$   
 $[uID.\ uID \leftarrow userIDs\ s,\ isChair\ s\ confID\ uID]$

**definition**  $e\text{-listChair} :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow bool$

**where**

$e\text{-listChair}\ s\ confID\ uID\ p \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [] \wedge pass\ s\ uID = p \wedge$   
 $(phase\ s\ confID \geq subPH \vee (phase\ s\ confID \geq setPH \wedge isChair\ s\ confID\ uID))$

## 2.2.6 Actions available in the subPH phase

**definition**  $createPaper :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow string \Rightarrow string \Rightarrow state$

**where**

```
createPaper s confID uID p papID title abstract ≡  
  let papIDs = paperIDs s confID;  
      rls = roles s confID uID  
  in  
  s (paperIDs := (paperIDs s) (confID := papID # papIDs),  
     paper := (paper s) (papID := Paper title abstract NoPContent [] (Dis [] [])),  
     roles := fun-upd2 (roles s) confID uID (List.insert (Aut papID) rls),  
     pref := fun-upd2 (pref s) uID papID Conflict)
```

**definition**  $e\text{-createPaper} :: \text{state} \Rightarrow \text{confID} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{paperID} \Rightarrow \text{string} \Rightarrow \text{string} \Rightarrow \text{bool}$

**where**

```
e-createPaper s confID uID p papID name info ≡  
  IDsOK s [confID] [uID] [] ∧ pass s uID = p ∧  
  phase s confID = subPH ∧  
  ¬ papID ∈∈ getAllPaperIDs s
```

**definition**  $\text{createAuthor} :: \text{state} \Rightarrow \text{confID} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{paperID} \Rightarrow \text{userID} \Rightarrow \text{state}$

**where**

```
createAuthor s confID uID p papID uID' ≡  
  let rls = roles s confID uID'  
  in  
  s (roles := fun-upd2 (roles s) confID uID' (List.insert (Aut papID) rls),  
     pref := fun-upd2 (pref s) uID' papID Conflict)
```

**definition**  $e\text{-createAuthor} :: \text{state} \Rightarrow \text{confID} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{paperID} \Rightarrow \text{userID} \Rightarrow \text{bool}$

**where**

```
e-createAuthor s confID uID p papID uID' ≡  
  IDsOK s [confID] [uID, uID'] [papID] ∧ pass s uID = p ∧  
  phase s confID = subPH ∧ isAut s confID uID papID ∧ uID ≠ uID'
```

**definition**  $\text{updatePaperTA} :: \text{state} \Rightarrow \text{confID} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{paperID} \Rightarrow \text{string} \Rightarrow \text{string} \Rightarrow \text{state}$

**where**

```
updatePaperTA s confID uID p papID title abstract ≡  
  case paper s papID of Paper title' abstract' pc reviews dis decs ⇒  
  s (paper := (paper s) (papID := Paper title abstract pc reviews dis decs))
```

**definition**  $e\text{-updatePaperTA} :: \text{state} \Rightarrow \text{confID} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{paperID} \Rightarrow \text{string} \Rightarrow \text{string} \Rightarrow \text{bool}$

**where**

```
e-updatePaperTA s confID uID p papID name info ≡
```

$IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$   
 $phase\ s\ confID = subPH \wedge isAut\ s\ confID\ uID\ papID$

**definition**  $updatePaperC :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow$   
 $pcontent \Rightarrow state$

**where**

$updatePaperC\ s\ confID\ uID\ p\ papID\ pc \equiv$   
 $case\ paper\ s\ papID\ of\ Paper\ title\ abstract\ pc'\ reviews\ dis\ decs \Rightarrow$   
 $s\ (\langle paper := (paper\ s)\ (papID := Paper\ title\ abstract\ pc\ reviews\ dis\ decs) \rangle)$

**definition**  $e-updatePaperC :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID$   
 $\Rightarrow pcontent \Rightarrow bool$

**where**

$e-updatePaperC\ s\ confID\ uID\ p\ papID\ pc \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$   
 $phase\ s\ confID = subPH \wedge isAut\ s\ confID\ uID\ papID$

**definition**  $createConflict :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow$   
 $userID \Rightarrow state$

**where**

$createConflict\ s\ confID\ uID\ p\ papID\ uID' \equiv$   
 $s\ (\langle pref := fun-upd2\ (pref\ s)\ uID'\ papID\ Conflict \rangle)$

**definition**  $e-createConflict :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID$   
 $\Rightarrow userID \Rightarrow bool$

**where**

$e-createConflict\ s\ confID\ uID\ p\ papID\ uID' \equiv$   
 $IDsOK\ s\ [confID]\ [uID, uID']\ [papID] \wedge pass\ s\ uID = p \wedge$   
 $phase\ s\ confID = subPH \wedge isAut\ s\ confID\ uID\ papID \wedge isPC\ s\ confID\ uID'$

## 2.2.7 Actions available starting from the subPH phase

**definition**  $readPaperTAA :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow$   
 $(string * string * userID\ list)$

**where**

$readPaperTAA\ s\ confID\ uID\ p\ papID \equiv$   
 $case\ paper\ s\ papID\ of\ Paper\ title\ abstract\ pc\ reviews\ dis\ decs \Rightarrow$   
 $(title, abstract, [uID.\ uID \leftarrow userIDs\ s, isAut\ s\ confID\ uID\ papID])$

**definition**  $e-readPaperTAA :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID$   
 $\Rightarrow bool$

**where**

$e-readPaperTAA\ s\ confID\ uID\ p\ papID \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$   
 $phase\ s\ confID \geq subPH \wedge (isAut\ s\ confID\ uID\ papID \vee isPC\ s\ confID\ uID)$

**definition**  $readPaperC :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow pcontent$

**where**

$readPaperC\ s\ confID\ uID\ p\ papID \equiv$   
 $case\ paper\ s\ papID\ of\ Paper\ title\ abstract\ pc\ reviews\ dis\ decs \Rightarrow pc$

**definition**  $e-readPaperC :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow bool$

**where**

$e-readPaperC\ s\ confID\ uID\ p\ papID \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge\ pass\ s\ uID = p \wedge$   
 $($   
 $phase\ s\ confID \geq\ subPH \wedge\ isAut\ s\ confID\ uID\ papID \vee$   
 $phase\ s\ confID \geq\ bidPH \wedge\ isPC\ s\ confID\ uID$   
 $)$

**definition**  $listPapers :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID\ list$

**where**

$listPapers\ s\ confID\ uID\ p \equiv$   
 $let\ paps = paper\ s\ in$   
 $[papID.\ papID \leftarrow paperIDs\ s\ confID]$

**definition**  $e-listPapers :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow bool$

**where**

$e-listPapers\ s\ confID\ uID\ p \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [] \wedge\ pass\ s\ uID = p \wedge$   
 $phase\ s\ confID \geq\ subPH \wedge\ isPC\ s\ confID\ uID$

**definition**  $listMyPapers :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID\ list$

**where**

$listMyPapers\ s\ confID\ uID\ p \equiv$   
 $let\ paps = paper\ s\ in$   
 $[papID.\ papID \leftarrow paperIDs\ s\ confID,\ isAut\ s\ confID\ uID\ papID]$

**definition**  $e-listMyPapers :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow bool$

**where**

$e-listMyPapers\ s\ confID\ uID\ p \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [] \wedge\ pass\ s\ uID = p \wedge$   
 $phase\ s\ confID \geq\ subPH$

## 2.2.8 Actions available in the bidPH phase

**definition**  $updatePref :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow preference \Rightarrow state$

**where**

$updatePref\ s\ confID\ uID\ p\ papID\ pr \equiv$   
 $s\ (\backslash pref := fun-upd2\ (pref\ s)\ uID\ papID\ pr)$

**definition**  $e-updatePref :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow preference \Rightarrow bool$

**where**

$e-updatePref\ s\ confID\ uID\ p\ papID\ pr \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$   
 $phase\ s\ confID = bidPH \wedge isPC\ s\ confID\ uID \wedge$   
 $\neg isAut\ s\ confID\ uID\ papID$

### 2.2.9 Actions available starting from the bidPH phase

**definition**  $readPref :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow preference$

**where**

$readPref\ s\ confID\ uID\ p\ papID \equiv$   
 $pref\ s\ uID\ papID$

**definition**  $e-readPref :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow bool$

**where**

$e-readPref\ s\ confID\ uID\ p\ papID \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$   
 $phase\ s\ confID \geq bidPH \wedge isPC\ s\ confID\ uID$

### 2.2.10 Actions available in the revPH phase

**definition**  $readPrefOfPC :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow userID \Rightarrow preference$

**where**

$readPrefOfPC\ s\ confID\ uID\ p\ papID\ uID' \equiv$   
 $pref\ s\ uID'\ papID$

**definition**  $e-readPrefOfPC :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow userID \Rightarrow bool$

**where**

$e-readPrefOfPC\ s\ confID\ uID\ p\ papID\ uID' \equiv$   
 $IDsOK\ s\ [confID]\ [uID, uID']\ [papID] \wedge pass\ s\ uID = p \wedge$   
 $(phase\ s\ confID \geq bidPH \wedge isChair\ s\ confID\ uID \wedge isPC\ s\ confID\ uID'$   
 $\vee$   
 $phase\ s\ confID = subPH \wedge isAut\ s\ confID\ uID\ papID)$

**definition**  $createReview :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow userID \Rightarrow state$

**where**

$createReview\ s\ confID\ uID\ p\ papID\ uID' \equiv$   
 $case\ paper\ s\ papID\ of\ Paper\ title\ abstract\ pc\ reviews\ dis\ decs \Rightarrow$   
 $let\ rls = roles\ s\ confID\ uID';\ n = length\ (reviewsPaper\ (paper\ s\ papID));$

$reviews' = reviews @ [emptyReview]$   
*in*  
 $s (roles := fun-upd2 (roles s) confID uID' (List.insert (Rev papID n) rls),$   
 $paper := fun-upd (paper s) papID (Paper title abstract pc reviews' dis decs)$   
 $)$

**definition**  $e-createReview :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow userID \Rightarrow bool$

**where**

$e-createReview s confID uID p papID uID' \equiv$   
 $IDsOK s [confID] [uID, uID'] [papID] \wedge pass s uID = p \wedge$   
 $phase s confID = revPH \wedge$   
 $isChair s confID uID \wedge pref s uID papID \neq Conflict \wedge$   
 $isPC s confID uID' \wedge \neg isRev s confID uID' papID \wedge pref s uID' papID \neq Conflict$

**definition**  $updateReview ::$

$state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow nat \Rightarrow rcontent \Rightarrow state$

**where**

$updateReview s confID uID p papID n rc \equiv$   
 $case\ paper\ s\ papID\ of\ Paper\ title\ abstract\ pc\ reviews\ dis\ decs \Rightarrow$   
 $let\ review = [rc];\ reviews' = list-update\ reviews\ n\ review$   
*in*  
 $s (paper := fun-upd (paper s) papID (Paper title abstract pc reviews' dis decs))$

**definition**  $e-updateReview ::$

$state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow nat \Rightarrow rcontent \Rightarrow bool$

**where**

$e-updateReview s confID uID p papID n rc \equiv$   
 $IDsOK s [confID] [uID] [papID] \wedge pass s uID = p \wedge$   
 $phase s confID = revPH \wedge isRev s confID uID papID \wedge$   
 $getReviewIndex s confID uID papID = n$

### 2.2.11 Actions available starting from the revPH phase

**definition**  $readMyReview :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow nat * review$

**where**

$readMyReview s confID uID p papID \equiv$   
 $case\ getRevRole\ s\ confID\ uID\ papID\ of$   
 $Some (Rev papID' n) \Rightarrow (n, getNthReview s papID n)$

**definition**  $e-readMyReview :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow bool$

**where**

$e-readMyReview s confID uID p papID \equiv$   
 $IDsOK s [confID] [uID] [papID] \wedge pass s uID = p \wedge$   
 $phase s confID \geq revPH \wedge isRev s confID uID papID$

**definition**  $listMyAssignedPapers :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow list$

**where**

$listMyAssignedPapers\ s\ confID\ uID\ p \equiv$   
 $let\ paps = paper\ s\ in$   
 $[papID.\ papID \leftarrow paperIDs\ s\ confID,\ isRev\ s\ confID\ uID\ papID]$

**definition**  $e-listMyAssignedPapers :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow bool$

**where**

$e-listMyAssignedPapers\ s\ confID\ uID\ p \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [] \wedge pass\ s\ uID = p \wedge$   
 $phase\ s\ confID \geq revPH \wedge isPC\ s\ confID\ uID$

**definition**  $listAssignedReviewers :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow userID \Rightarrow list$

**where**

$listAssignedReviewers\ s\ confID\ uID\ p\ papID \equiv$   
 $[uID \leftarrow userIDs\ s.\ isRev\ s\ confID\ uID\ papID]$

**definition**  $e-listAssignedReviewers :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow bool$

**where**

$e-listAssignedReviewers\ s\ confID\ uID\ p\ papID \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$   
 $phase\ s\ confID \geq revPH \wedge$   
 $isChair\ s\ confID\ uID \wedge pref\ s\ uID\ papID \neq Conflict$

## 2.2.12 Actions available in the disPH phase

**definition**  $uupdateDis :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow string \Rightarrow state$

**where**

$uupdateDis\ s\ confID\ uID\ p\ papID\ comm \equiv$   
 $case\ paper\ s\ papID\ of\ Paper\ title\ abstract\ pc\ reviews\ (Dis\ comments)\ decs \Rightarrow$   
 $s\ (\backslash paper := fun-upd\ (paper\ s)\ papID\ (Paper\ title\ abstract\ pc\ reviews\ (Dis\ (comm\ \# comments))\ decs))$

**definition**  $e-uupdateDis :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow string \Rightarrow bool$

**where**

$e-uupdateDis\ s\ confID\ uID\ p\ papID\ comm \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$   
 $phase\ s\ confID = disPH \wedge isPC\ s\ confID\ uID \wedge pref\ s\ uID\ papID \neq Conflict$

**definition** *updateReview* ::

$state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow nat \Rightarrow rcontent \Rightarrow state$

**where**

$updateReview\ s\ confID\ uID\ p\ papID\ n\ rc \equiv$

$case\ paper\ s\ papID\ of\ Paper\ title\ abstract\ pc\ reviews\ dis\ decs \Rightarrow$

$let\ review = rc \# (reviews ! n); reviews' = list-update\ reviews\ n\ review$

$in$

$s\ (\!paper := fun-upd\ (paper\ s)\ papID\ (Paper\ title\ abstract\ pc\ reviews'\ dis\ decs)\!)$

**definition** *e-updateReview* ::

$state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow nat \Rightarrow rcontent \Rightarrow bool$

**where**

$e-updateReview\ s\ confID\ uID\ p\ papID\ n\ rc \equiv$

$IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$

$phase\ s\ confID = disPH \wedge isRev\ s\ confID\ uID\ papID \wedge$

$getReviewIndex\ s\ confID\ uID\ papID = n$

**definition** *updateDec* ::  $state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow$

$decision \Rightarrow state$

**where**

$updateDec\ s\ confID\ uID\ p\ papID\ dec \equiv$

$case\ paper\ s\ papID\ of\ Paper\ title\ abstract\ pc\ reviews\ dis\ decs \Rightarrow$

$s\ (\!paper := fun-upd\ (paper\ s)\ papID\ (Paper\ title\ abstract\ pc\ reviews\ dis\ (dec\ \# decs)\!)$

**definition** *e-updateDec* ::  $state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow$

$decision \Rightarrow bool$

**where**

$e-updateDec\ s\ confID\ uID\ p\ papID\ dec \equiv$

$IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$

$phase\ s\ confID = disPH \wedge isChair\ s\ confID\ uID \wedge pref\ s\ uID\ papID \neq Conflict$

### 2.2.13 Actions available starting from the disPH phase

**definition** *readReviews* ::  $state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow$

$(userID * review)\ list$

**where**

$readReviews\ s\ confID\ uID\ p\ papID \equiv$

$getReviewersReviews\ s\ confID\ papID$

**definition** *e-readReviews* ::  $state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow$

$bool$

**where**

$e-readReviews\ s\ confID\ uID\ p\ papID \equiv$

$IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$

$phase\ s\ confID \geq disPH \wedge isPC\ s\ confID\ uID \wedge pref\ s\ uID\ papID \neq Conflict$

**definition**  $readDecs :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow decision\ list$

**where**

$readDecs\ s\ confID\ uID\ p\ papID \equiv$   
*case paper s papID of Paper title abstract pc reviews dis decs  $\Rightarrow$  decs*

**definition**  $e-readDecs :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow bool$

**where**

$e-readDecs\ s\ confID\ uID\ p\ papID \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$   
 $phase\ s\ confID \geq disPH \wedge isPC\ s\ confID\ uID \wedge pref\ s\ uID\ papID \neq Conflict$

**definition**  $readDis :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow string\ list$

**where**

$readDis\ s\ confID\ uID\ p\ papID \equiv$   
*case paper s papID of Paper title abstract pc reviews (Dis comments) decs  $\Rightarrow$  comments*

**definition**  $e-readDis :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow bool$

**where**

$e-readDis\ s\ confID\ uID\ p\ papID \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$   
 $phase\ s\ confID \geq disPH \wedge isPC\ s\ confID\ uID \wedge pref\ s\ uID\ papID \neq Conflict$

## 2.2.14 Actions available starting from the notifPH phase

**definition**  $readFinalReviews :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow review\ list$

**where**

$readFinalReviews\ s\ confID\ uID\ p\ papID \equiv$   
 $map\ (\lambda\ rev.\ case\ rev\ of\ [] \Rightarrow [(NoExp, NoScore, emptyStr)]$   
 $\quad | ((exp, score, comm) \# rv) \Rightarrow [(exp, score, comm)])$   
 $(reviewsPaper\ (paper\ s\ papID))$

**definition**  $e-readFinalReviews :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow bool$

**where**

$e-readFinalReviews\ s\ confID\ uID\ p\ papID \equiv$   
 $IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$   
 $phase\ s\ confID \geq notifPH \wedge (isAut\ s\ confID\ uID\ papID \vee (isPC\ s\ confID\ uID \wedge$   
 $pref\ s\ uID\ papID \neq Conflict))$

**definition**  $readFinalDec :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow decision$

**where**

$readFinalDec\ s\ confID\ uID\ p\ papID \equiv$

*case paper s papID of Paper title abstract pc reviews dis decs ⇒  
case decs of [] ⇒ NoDecision | dec # decs ⇒ dec*

**definition** *e-readFinalDec :: state ⇒ confID ⇒ userID ⇒ password ⇒ paperID  
⇒ bool*

**where**

*e-readFinalDec s confID uID p papID ≡  
IDsOK s [confID] [uID] [papID] ∧ pass s uID = p ∧  
phase s confID ≥ notifPH ∧ (isAut s confID uID papID ∨ isPC s confID uID)*

## 2.3 The step function

**datatype** *out =*

*outOK | outErr |  
outBool bool |  
outSTRT string \* string \* string | outSTRL string list | outCONF string \* string  
\* role list \* phase |  
outPREF preference |  
outCON pcontent |  
outNREV nat \* review | outREVL review list | outRREVL (userID \* review) list  
|  
outDEC decision | outDECL decision list |  
outCIDL confID list | outUIDL userID list | outPIDL paperID list |  
outSTRPAL string \* string \* userID list*

**datatype** *cAct =*

*cUser userID password string string string  
| cConf confID userID password string string  
| cPC confID userID password userID  
| cChair confID userID password userID  
| cPaper confID userID password paperID string string  
| cAuthor confID userID password paperID userID  
| cConflict confID userID password paperID userID  
| cReview confID userID password paperID userID*

**lemmas** *c-defs =*

*e-createUser-def createUser-def  
e-createConf-def createConf-def  
e-createPC-def createPC-def  
e-createChair-def createChair-def  
e-createAuthor-def createAuthor-def  
e-createConflict-def createConflict-def  
e-createPaper-def createPaper-def  
e-createReview-def createReview-def*

**fun** *cStep :: state ⇒ cAct ⇒ out \* state where*

*cStep s (cUser uID p name info email) =  
(if e-createUser s uID p name info email  
then (outOK, createUser s uID p name info email)*

```

    else (outErr, s))
|
cStep s (cConf confID uID p name info) =
  (if e-createConf s confID uID p name info
   then (outOK, createConf s confID uID p name info)
   else (outErr, s))
|
cStep s (cPC confID uID p uID') =
  (if e-createPC s confID uID p uID'
   then (outOK, createPC s confID uID p uID')
   else (outErr, s))
|
cStep s (cChair confID uID p uID') =
  (if e-createChair s confID uID p uID'
   then (outOK, createChair s confID uID p uID')
   else (outErr, s))
|
cStep s (cPaper confID uID p papID name info) =
  (if e-createPaper s confID uID p papID name info
   then (outOK, createPaper s confID uID p papID name info)
   else (outErr, s))
|
cStep s (cAuthor confID uID p papID uID') =
  (if e-createAuthor s confID uID p papID uID'
   then (outOK, createAuthor s confID uID p papID uID')
   else (outErr, s))
|
cStep s (cConflict confID uID p papID uID') =
  (if e-createConflict s confID uID p papID uID'
   then (outOK, createConflict s confID uID p papID uID')
   else (outErr, s))
|
cStep s (cReview confID uID p papID uID') =
  (if e-createReview s confID uID p papID uID'
   then (outOK, createReview s confID uID p papID uID')
   else (outErr, s))

```

**datatype** *uAct* =

```

  uUser userID password password string string string
|uConfA confID userID password
|uPhase confID userID password phase
|uPaperTA confID userID password paperID string string
|uPaperC confID userID password paperID pcontent
|uPref confID userID password paperID preference
|uReview confID userID password paperID nat rcontent

```

**lemmas** *u-defs* =

```

e-updateUser-def updateUser-def

```

*e-updateConfA-def updateConfA-def*  
*e-updatePhase-def updatePhase-def*  
*e-updatePaperTA-def updatePaperTA-def*  
*e-updatePaperC-def updatePaperC-def*  
*e-updatePref-def updatePref-def*  
*e-updateReview-def updateReview-def*

**fun** *uStep* :: *state* ⇒ *uAct* ⇒ *out* \* *state* **where**  
*uStep* *s* (*uUser* *uID* *p* *p'* *name* *info* *email*) =  
 (if *e-updateUser* *s* *uID* *p* *p'* *name* *info* *email*  
 then (*outOK*, *updateUser* *s* *uID* *p* *p'* *name* *info* *email*)  
 else (*outErr*, *s*))  
 |  
*uStep* *s* (*uConfA* *confID* *uID* *p*) =  
 (if *e-updateConfA* *s* *confID* *uID* *p*  
 then (*outOK*, *updateConfA* *s* *confID* *uID* *p*)  
 else (*outErr*, *s*))  
 |  
*uStep* *s* (*uPhase* *confID* *uID* *p* *ph*) =  
 (if *e-updatePhase* *s* *confID* *uID* *p* *ph*  
 then (*outOK*, *updatePhase* *s* *confID* *uID* *p* *ph*)  
 else (*outErr*, *s*))  
 |  
*uStep* *s* (*uPaperTA* *confID* *uID* *p* *papID* *name* *info*) =  
 (if *e-updatePaperTA* *s* *confID* *uID* *p* *papID* *name* *info*  
 then (*outOK*, *updatePaperTA* *s* *confID* *uID* *p* *papID* *name* *info*)  
 else (*outErr*, *s*))  
 |  
*uStep* *s* (*uPaperC* *confID* *uID* *p* *papID* *pc*) =  
 (if *e-updatePaperC* *s* *confID* *uID* *p* *papID* *pc*  
 then (*outOK*, *updatePaperC* *s* *confID* *uID* *p* *papID* *pc*)  
 else (*outErr*, *s*))  
 |  
*uStep* *s* (*uPref* *confID* *uID* *p* *papID* *pr*) =  
 (if *e-updatePref* *s* *confID* *uID* *p* *papID* *pr*  
 then (*outOK*, *updatePref* *s* *confID* *uID* *p* *papID* *pr*)  
 else (*outErr*, *s*))  
 |  
*uStep* *s* (*uReview* *confID* *uID* *p* *papID* *n* *rc*) =  
 (if *e-updateReview* *s* *confID* *uID* *p* *papID* *n* *rc*  
 then (*outOK*, *updateReview* *s* *confID* *uID* *p* *papID* *n* *rc*)  
 else (*outErr*, *s*))

**datatype** *uuAct* =  
*uuNews* *confID* *userID* *password* *string*  
 | *uuDis* *confID* *userID* *password* *paperID* *string*  
 | *uuReview* *confID* *userID* *password* *paperID* *nat* *rcontent*  
 | *uuDec* *confID* *userID* *password* *paperID* *decision*

```

lemmas uu-defs =
  e-uupdateNews-def uupdateNews-def
  e-uupdateDis-def uupdateDis-def
  e-uupdateReview-def uupdateReview-def
  uupdateDec-def e-uupdateDec-def

fun uuStep :: state ⇒ uuAct ⇒ out * state where
  uuStep s (uuNews confID uID p ev) =
    (if e-uupdateNews s confID uID p ev
     then (outOK, uupdateNews s confID uID p ev)
     else (outErr, s))
  |
  uuStep s (uuDis confID uID p papID comm) =
    (if e-uupdateDis s confID uID p papID comm
     then (outOK, uupdateDis s confID uID p papID comm)
     else (outErr, s))
  |
  uuStep s (uuReview confID uID p papID n rc) =
    (if e-uupdateReview s confID uID p papID n rc
     then (outOK, uupdateReview s confID uID p papID n rc)
     else (outErr, s))
  |
  uuStep s (uuDec confID uID p papID dec) =
    (if e-uupdateDec s confID uID p papID dec
     then (outOK, uupdateDec s confID uID p papID dec)
     else (outErr, s))

```

```

datatype rAct =
  rAmIVoronkov userID password
  | rUser userID password userID
  | rConf confID userID password
  | rNews confID userID password
  | rPaperTAA confID userID password paperID
  | rPaperC confID userID password paperID
  | rPref confID userID password paperID
  | rMyReview confID userID password paperID
  | rReviews confID userID password paperID
  | rDecs confID userID password paperID
  | rDis confID userID password paperID
  | rFinalReviews confID userID password paperID
  | rFinalDec confID userID password paperID
  | rPrefOfPC confID userID password paperID userID

```

```

lemmas r-defs =
  readAmIVoronkov-def e-readAmIVoronkov-def
  readUser-def e-readUser-def
  readConf-def e-readConf-def
  readNews-def e-readNews-def

```

*readPaperTAA-def e-readPaperTAA-def*  
*readPaperC-def e-readPaperC-def*  
*readPref-def e-readPref-def*  
*readMyReview-def e-readMyReview-def*  
*readReviews-def e-readReviews-def*  
*readDecs-def e-readDecs-def*  
*readDis-def e-readDis-def*  
*readFinalReviews-def e-readFinalReviews-def*  
*readFinalDec-def e-readFinalDec-def*  
*readPrefOfPC-def e-readPrefOfPC-def*

**fun** *rObs* :: *state*  $\Rightarrow$  *rAct*  $\Rightarrow$  *out* **where**  
*rObs s (rAmIVoronkov uID p) =*  
   (*if e-readAmIVoronkov s uID p then outBool (readAmIVoronkov s uID p) else outErr*)  
 |  
*rObs s (rUser uID p uID') =*  
   (*if e-readUser s uID p uID' then outSTRT (readUser s uID p uID') else outErr*)  
 |  
*rObs s (rConf confID uID p) =*  
   (*if e-readConf s confID uID p then outCONF (readConf s confID uID p) else outErr*)  
 |  
*rObs s (rNews confID uID p) =*  
   (*if e-readNews s confID uID p then outSTRL (readNews s confID uID p) else outErr*)  
 |  
*rObs s (rPaperTAA confID uID p papID) =*  
   (*if e-readPaperTAA s confID uID p papID then outSTRPAL (readPaperTAA s confID uID p papID) else outErr*)  
 |  
*rObs s (rPaperC confID uID p papID) =*  
   (*if e-readPaperC s confID uID p papID then outCON (readPaperC s confID uID p papID) else outErr*)  
 |  
*rObs s (rPref confID uID p papID) =*  
   (*if e-readPref s confID uID p papID then outPREF (readPref s confID uID p papID) else outErr*)  
 |  
*rObs s (rMyReview confID uID p papID) =*  
   (*if e-readMyReview s confID uID p papID then outNREV (readMyReview s confID uID p papID) else outErr*)  
 |  
*rObs s (rReviews confID uID p papID) =*  
   (*if e-readReviews s confID uID p papID then outRREVL (readReviews s confID uID p papID) else outErr*)  
 |  
*rObs s (rDecs confID uID p papID) =*  
   (*if e-readDecs s confID uID p papID then outDECL (readDecs s confID uID p*

```

papID) else outErr)
|
rObs s (rDis confID uID p papID) =
  (if e-readDis s confID uID p papID then outSTRL (readDis s confID uID p papID)
  else outErr)
|
rObs s (rFinalReviews confID uID p papID) =
  (if e-readFinalReviews s confID uID p papID then outREVL (readFinalReviews s
  confID uID p papID) else outErr)
|
rObs s (rFinalDec confID uID p papID) =
  (if e-readFinalDec s confID uID p papID then outDEC (readFinalDec s confID
  uID p papID) else outErr)
|
rObs s (rPrefOfPC confID uID p papID uID') =
  (if e-readPrefOfPC s confID uID p papID uID' then outPREF (readPrefOfPC s
  confID uID p papID uID') else outErr)

```

```

datatype lAct =
  lConfs userID password
| lAConfs userID password
| lSConfs userID password
| lMyConfs userID password
| lAllUsers userID password
| lAllPapers userID password
| lPC confID userID password
| lChair confID userID password
| lPapers confID userID password
| lMyPapers confID userID password
| lMyAssignedPapers confID userID password
| lAssignedReviewers confID userID password paperID

```

```

lemmas l-defs =
  listConfs-def e-listConfs-def
  listAConfs-def e-listAConfs-def
  listSConfs-def e-listSConfs-def
  listMyConfs-def e-listMyConfs-def
  listAllUsers-def e-listAllUsers-def
  listAllPapers-def e-listAllPapers-def
  listPC-def e-listPC-def
  listChair-def e-listChair-def
  listPapers-def e-listPapers-def
  listMyPapers-def e-listMyPapers-def
  listMyAssignedPapers-def e-listMyAssignedPapers-def
  listAssignedReviewers-def e-listAssignedReviewers-def

```

```

fun lObs :: state ⇒ lAct ⇒ out where
lObs s (lConfs uID p) =
  (if e-listConfs s uID p then outCIDL (listConfs s uID p) else outErr)

```

```

|
lObs s (lAConfs uID p) =
  (if e-listAConfs s uID p then outCIDL (listAConfs s uID p) else outErr)
|
lObs s (lSConfs uID p) =
  (if e-listSConfs s uID p then outCIDL (listSConfs s uID p) else outErr)
|
lObs s (lMyConfs uID p) =
  (if e-listMyConfs s uID p then outCIDL (listMyConfs s uID p) else outErr)
|
lObs s (lAllUsers uID p) =
  (if e-listAllUsers s uID p then outUIDL (listAllUsers s uID p) else outErr)
|
lObs s (lAllPapers uID p) =
  (if e-listAllPapers s uID p then outPIDL (listAllPapers s uID p) else outErr)
|
lObs s (lPC confID uID p) =
  (if e-listPC s confID uID p then outUIDL (listPC s confID uID p) else outErr)
|
lObs s (lChair confID uID p) =
  (if e-listChair s confID uID p then outUIDL (listChair s confID uID p) else outErr)
|
lObs s (lPapers confID uID p) =
  (if e-listPapers s confID uID p then outPIDL (listPapers s confID uID p) else
  outErr)
|
lObs s (lMyPapers confID uID p) =
  (if e-listMyPapers s confID uID p then outPIDL (listMyPapers s confID uID p)
  else outErr)
|
lObs s (lMyAssignedPapers confID uID p) =
  (if e-listMyAssignedPapers s confID uID p then outPIDL (listMyAssignedPapers s
  confID uID p) else outErr)
|
lObs s (lAssignedReviewers confID uID p papID) =
  (if e-listAssignedReviewers s confID uID p papID
  then outUIDL (listAssignedReviewers s confID uID p papID) else outErr)

```

**datatype** *act* =

*Cact* *cAct* | *Uact* *uAct* | *UUact* *uuAct* |

*Ract* *rAct* | *Lact* *lAct*

**fun** *step* :: *state* ⇒ *act* ⇒ *out* \* *state* **where**

*step* *s* (*Cact* *ca*) = *cStep* *s* *ca*

*step* *s* (*Uact* *ua*) = *uStep* *s* *ua*

```

step s (UUact uua) = uuStep s uua
|
step s (Ract ra) = (rObs s ra, s)
|
step s (Lact la) = (lObs s la, s)

```

**export-code** step istate getFreshPaperID in Scala

Some action selectors, used for verification:

```

fun cUserOfA :: cAct ⇒ userID where
cUserOfA (cUser uID p name info email) = uID
|
cUserOfA (cConf confID uID p name info) = uID
|
cUserOfA (cPC confID uID p uID') = uID
|
cUserOfA (cChair confID uID p uID') = uID
|
cUserOfA (cPaper confID uID p papID name info) = uID
|
cUserOfA (cAuthor confID uID p papID uID') = uID
|
cUserOfA (cConflict confID uID p papID uID') = uID
|
cUserOfA (cReview confID uID p papID uID') = uID

```

```

fun uUserOfA :: uAct ⇒ userID where
uUserOfA (uUser uID p p' name info email) = uID
|
uUserOfA (uConfA confID uID p) = uID
|
uUserOfA (uPhase confID uID p ph) = uID
|
uUserOfA (uPaperTA confID uID p papID name info) = uID
|
uUserOfA (uPaperC confID uID p papID pc) = uID
|
uUserOfA (uPref confID uID p papID pr) = uID
|
uUserOfA (uReview confID uID p papID n rc) = uID

```

```

fun uuUserOfA :: uuAct ⇒ userID where
uuUserOfA (uuNews confID uID p ev) = uID
|
uuUserOfA (uuDis confID uID p papID comm) = uID
|
uuUserOfA (uuReview confID uID p papID n rc) = uID
|
uuUserOfA (uuDec confID uID p papID dec) = uID

```

```

fun rUserOfA :: rAct ⇒ userID where
rUserOfA (rAmIVoronkov uID p) = uID
|
rUserOfA (rUser uID p uID') = uID
|
rUserOfA (rConf confID uID p) = uID
|
rUserOfA (rNews confID uID p) = uID
|
rUserOfA (rPaperTAA confID uID p papID) = uID
|
rUserOfA (rPaperC confID uID p papID) = uID
|
rUserOfA (rPref confID uID p papID) = uID
|
rUserOfA (rMyReview confID uID p papID) = uID
|
rUserOfA (rReviews confID uID p papID) = uID
|
rUserOfA (rDecs confID uID p papID) = uID
|
rUserOfA (rDis confID uID p papID) = uID
|
rUserOfA (rFinalReviews confID uID p papID) = uID
|
rUserOfA (rFinalDec confID uID p papID) = uID
|
rUserOfA (rPrefOfPC confID uID p papID uID') = uID

```

```

fun lUserOfA :: lAct ⇒ userID where
lUserOfA (lConfs uID p) = uID
|
lUserOfA (lAConfs uID p) = uID
|
lUserOfA (lSConfs uID p) = uID
|
lUserOfA (lMyConfs uID p) = uID
|
lUserOfA (lAllUsers uID p) = uID
|
lUserOfA (lAllPapers uID p) = uID
|
lUserOfA (lPC confID uID p) = uID
|
lUserOfA (lChair confID uID p) = uID
|
lUserOfA (lPapers confID uID p) = uID
|

```

```

lUserOfA (lMyPapers confID uID p) = uID
|
lUserOfA (lMyAssignedPapers confID uID p) = uID
|
lUserOfA (lAssignedReviewers confID uID p papID) = uID

```

```

fun userOfA :: act ⇒ userID where
userOfA (Cact ca) = cUserOfA ca
|
userOfA (Uact ua) = uUserOfA ua
|
userOfA (UUact uua) = uuUserOfA uua
|
userOfA (Ract ra) = rUserOfA ra
|
userOfA (Lact la) = lUserOfA la

```

```

fun cConfOfA :: cAct ⇒ confID option where
cConfOfA (cUser uID p name info email) = None
|
cConfOfA (cConf confID uID p name info) = Some confID
|
cConfOfA (cPC confID uID p uID') = Some confID
|
cConfOfA (cChair confID uID p uID') = Some confID
|
cConfOfA (cPaper confID uID p papID name info) = Some confID
|
cConfOfA (cAuthor confID uID p papID uID') = Some confID
|
cConfOfA (cConflict confID uID p papID uID') = Some confID
|
cConfOfA (cReview confID uID p papID uID') = Some confID

```

```

fun uConfOfA :: uAct ⇒ confID option where
uConfOfA (uUser uID p p' name info email) = None
|
uConfOfA (uConfA confID uID p) = Some confID
|
uConfOfA (uPhase confID uID p ph) = Some confID
|
uConfOfA (uPaperTA confID uID p papID name info) = Some confID
|
uConfOfA (uPaperC confID uID p papID pc) = Some confID
|
uConfOfA (uPref confID uID p papID pr) = Some confID

```

```

|
uConfOfA (uReview confID uID p papID n rc) = Some confID

fun uuConfOfA :: uuAct ⇒ confID option where
uuConfOfA (uuNews confID uID p ev) = Some confID
|
uuConfOfA (uuDis confID uID p papID comm) = Some confID
|
uuConfOfA (uuReview confID uID p papID n rc) = Some confID
|
uuConfOfA (uuDec confID uID p papID dec) = Some confID

fun rConfOfA :: rAct ⇒ confID option where
rConfOfA (rAmIVoronkov uID p) = None
|
rConfOfA (rUser uID p uID') = None
|
rConfOfA (rConf confID uID p) = Some confID
|
rConfOfA (rNews confID uID p) = Some confID
|
rConfOfA (rPaperTAA confID uID p papID) = Some confID
|
rConfOfA (rPaperC confID uID p papID) = Some confID
|
rConfOfA (rPref confID uID p papID) = Some confID
|
rConfOfA (rMyReview confID uID p papID) = Some confID
|
rConfOfA (rReviews confID uID p papID) = Some confID
|
rConfOfA (rDecs confID uID p papID) = Some confID
|
rConfOfA (rDis confID uID p papID) = Some confID
|
rConfOfA (rFinalReviews confID uID p papID) = Some confID
|
rConfOfA (rFinalDec confID uID p papID) = Some confID
|
rConfOfA (rPrefOfPC confID uID p papID uID') = Some confID

fun lConfOfA :: lAct ⇒ confID option where
lConfOfA (lConfs uID p) = None
|
lConfOfA (lAConfs uID p) = None
|
lConfOfA (lSConfs uID p) = None
|

```

```

lConfOfA (lMyConfs uID p) = None
|
lConfOfA (lAllUsers uID p) = None
|
lConfOfA (lAllPapers uID p) = None
|
lConfOfA (lPC confID uID p) = Some confID
|
lConfOfA (lChair confID uID p) = Some confID
|
lConfOfA (lPapers confID uID p) = Some confID
|
lConfOfA (lMyPapers confID uID p) = Some confID
|
lConfOfA (lMyAssignedPapers confID uID p) = Some confID
|
lConfOfA (lAssignedReviewers confID uID p papID) = Some confID

fun confOfA :: act ⇒ confID option where
confOfA (Cact ca) = cConfOfA ca
|
confOfA (Uact ua) = uConfOfA ua
|
confOfA (UUact uua) = uuConfOfA uua
|
confOfA (Ract ra) = rConfOfA ra
|
confOfA (Lact la) = lConfOfA la

fun cPaperOfA :: cAct ⇒ paperID option where
cPaperOfA (cUser uID p name info email) = None
|
cPaperOfA (cPaper confID uID p papID name info) = Some papID
|
cPaperOfA (cPC confID uID p uID') = None
|
cPaperOfA (cChair confID uID p uID') = None
|
cPaperOfA (cConf confID uID p name info) = None
|
cPaperOfA (cAuthor confID uID p papID uID') = Some papID
|
cPaperOfA (cConflict confID uID p papID uID') = Some papID
|
cPaperOfA (cReview confID uID p papID uID') = Some papID

```

```

fun uPaperOfA :: uAct ⇒ paperID option where
uPaperOfA (uUser uID p p' name info email) = None
|
uPaperOfA (uConfA confID uID p) = None
|
uPaperOfA (uPhase confID uID p ph) = None
|
uPaperOfA (uPaperTA confID uID p papID name info) = Some papID
|
uPaperOfA (uPaperC confID uID p papID pc) = Some papID
|
uPaperOfA (uPref confID uID p papID pr) = Some papID
|
uPaperOfA (uReview confID uID p papID n rc) = Some papID

```

```

fun uuPaperOfA :: uuAct ⇒ paperID option where
uuPaperOfA (uuNews confID uID p ev) = None
|
uuPaperOfA (uuDis confID uID p papID comm) = Some papID
|
uuPaperOfA (uuReview confID uID p papID n rc) = Some papID
|
uuPaperOfA (uuDec confID uID p papID dec) = Some papID

```

```

fun rPaperOfA :: rAct ⇒ paperID option where
rPaperOfA (rAmIVoronkov uID p) = None
|
rPaperOfA (rUser uID p uID') = None
|
rPaperOfA (rConf confID uID p) = None
|
rPaperOfA (rNews confID uID p) = None
|
rPaperOfA (rPaperTAA confID uID p papID) = Some papID
|
rPaperOfA (rPaperC confID uID p papID) = Some papID
|
rPaperOfA (rPref confID uID p papID) = Some papID
|
rPaperOfA (rMyReview confID uID p papID) = Some papID
|
rPaperOfA (rReviews confID uID p papID) = Some papID
|
rPaperOfA (rDecs confID uID p papID) = Some papID
|
rPaperOfA (rDis confID uID p papID) = Some papID
|
rPaperOfA (rFinalReviews confID uID p papID) = Some papID
|

```

```

rPaperOfA (rFinalDec confID uID p papID) = Some papID
|
rPaperOfA (rPrefOfPC confID uID p papID uID') = Some papID

fun lPaperOfA :: lAct ⇒ paperID option where
lPaperOfA (lConfs uID p) = None
|
lPaperOfA (lAConfs uID p) = None
|
lPaperOfA (lSConfs uID p) = None
|
lPaperOfA (lMyConfs uID p) = None
|
lPaperOfA (lAllUsers uID p) = None
|
lPaperOfA (lAllPapers uID p) = None
|
lPaperOfA (lPC confID uID p) = None
|
lPaperOfA (lChair confID uID p) = None
|
lPaperOfA (lPapers confID uID p) = None
|
lPaperOfA (lMyPapers confID uID p) = None
|
lPaperOfA (lMyAssignedPapers confID uID p) = None
|
lPaperOfA (lAssignedReviewers confID uID p papID) = Some papID

fun paperOfA :: act ⇒ paperID option where
paperOfA (Cact ca) = cPaperOfA ca
|
paperOfA (Uact ua) = uPaperOfA ua
|
paperOfA (UUact uua) = uuPaperOfA uua
|
paperOfA (Ract ra) = rPaperOfA ra
|
paperOfA (Lact la) = lPaperOfA la

```

```

end
theory Automation-Setup
imports System-Specification
begin

```

**lemma** *add-prop*:  
**assumes** *PROP* (*T*)  
**shows**  $A \implies PROP$  (*T*)  
**using** *assms* .

**lemmas** *exhaust-elim* =  
*cAct.exhaust*[of *x*, *THEN add-prop*[**where**  $A=a=Cact$  *x*], *rotated -1*]  
*uAct.exhaust*[of *x*, *THEN add-prop*[**where**  $A=a=Uact$  *x*], *rotated -1*]  
*uuAct.exhaust*[of *x*, *THEN add-prop*[**where**  $A=a=UUact$  *x*], *rotated -1*]  
*rAct.exhaust*[of *x*, *THEN add-prop*[**where**  $A=a=Ract$  *x*], *rotated -1*]  
*lAct.exhaust*[of *x*, *THEN add-prop*[**where**  $A=a=Lact$  *x*], *rotated -1*]  
**for** *x a*

**lemma** *Paper-dest-conv*:  
( $p =$   
*Paper title abstract content reviews dis decs*)  $\longleftrightarrow$   
*title = titlePaper p*  $\wedge$   
*abstract = abstractPaper p*  $\wedge$   
*content = contentPaper p*  $\wedge$   
*reviews = reviewsPaper p*  $\wedge$   
*dis = disPaper p*  $\wedge$   
*decs = decsPaper p*  
**by** (*cases p*) *auto*

**end**

### 3 Safety properties

**theory** *Safety-Properties*  
**imports** *Automation-Setup Bounded-Deducibility-Security.IO-Automaton*  
**begin**

**interpretation** *IO-Automaton* **where**  
*istate = istate* **and** *step = step*  
**done**

#### 3.1 Infrastructure for invariance reasoning

**definition** *cIsInvar* :: (*state*  $\Rightarrow$  *bool*)  $\Rightarrow$  *bool* **where**  
*cIsInvar*  $\varphi \equiv \forall s ca. reach\ s \wedge \varphi\ s \longrightarrow \varphi\ (snd\ (cStep\ s\ ca))$

**definition** *uIsInvar* :: (*state*  $\Rightarrow$  *bool*)  $\Rightarrow$  *bool* **where**  
*uIsInvar*  $\varphi \equiv \forall s ua. reach\ s \wedge \varphi\ s \longrightarrow \varphi\ (snd\ (uStep\ s\ ua))$

**definition**  $uuIsInvar :: (state \Rightarrow bool) \Rightarrow bool$  **where**  
 $uuIsInvar \varphi \equiv \forall s uua. reach\ s \wedge \varphi\ s \longrightarrow \varphi\ (snd\ (uuStep\ s\ uua))$

**lemma**  $invar\ cIsInvar\ uIsInvar\ uuIsInvar$ :  
 $invar\ \varphi \longleftrightarrow cIsInvar\ \varphi \wedge uIsInvar\ \varphi \wedge uuIsInvar\ \varphi$  (**is**  $?L \longleftrightarrow ?R$ )  
**unfolding**  $invar\ def\ cIsInvar\ def\ uIsInvar\ def\ uuIsInvar\ def\ fun\ eq\ iff$   
**apply**  $standard$   
**apply**  $(metis\ snd\ eqD\ step.simps)$   
**apply**  $safe$   
**subgoal for - a apply**( $cases\ a, auto$ ) .  
**done**

**lemma**  $cIsInvar[case\ names\ cUser\ cConf\ cPC\ cChair\ cPaper\ cAuthor\ cConflict\ cReview]$ :

**assumes**

$\bigwedge s\ uID\ p\ name\ info\ email.$

$\llbracket reach\ s; \varphi\ s; e\ createUser\ s\ uID\ p\ name\ info\ email \rrbracket$   
 $\implies \varphi\ (createUser\ s\ uID\ p\ name\ info\ email)$

**and**

$\bigwedge s\ confID\ uID\ p\ name\ info.$

$\llbracket reach\ s; \varphi\ s; e\ createConf\ s\ confID\ uID\ p\ name\ info \rrbracket$   
 $\implies \varphi\ (createConf\ s\ confID\ uID\ p\ name\ info)$

**and**

$\bigwedge s\ confID\ uID\ p\ uID'.$

$\llbracket reach\ s; \varphi\ s; e\ createPC\ s\ confID\ uID\ p\ uID' \rrbracket$   
 $\implies \varphi\ (createPC\ s\ confID\ uID\ p\ uID')$

**and**

$\bigwedge s\ confID\ uID\ p\ uID'.$

$\llbracket reach\ s; \varphi\ s; e\ createChair\ s\ confID\ uID\ p\ uID' \rrbracket$   
 $\implies \varphi\ (createChair\ s\ confID\ uID\ p\ uID')$

**and**

$\bigwedge s\ confID\ uID\ p\ papID\ name\ info.$

$\llbracket reach\ s; \varphi\ s; e\ createPaper\ s\ confID\ uID\ p\ papID\ name\ info \rrbracket$   
 $\implies \varphi\ (createPaper\ s\ confID\ uID\ p\ papID\ name\ info)$

**and**

$\bigwedge s\ confID\ uID\ p\ papID\ uID'.$

$\llbracket reach\ s; \varphi\ s; e\ createAuthor\ s\ confID\ uID\ p\ papID\ uID' \rrbracket$   
 $\implies \varphi\ (createAuthor\ s\ confID\ uID\ p\ papID\ uID')$

**and**

$\bigwedge s\ confID\ uID\ p\ papID\ uID'.$

$\llbracket reach\ s; \varphi\ s; e\ createConflict\ s\ confID\ uID\ p\ papID\ uID' \rrbracket$   
 $\implies \varphi\ (createConflict\ s\ confID\ uID\ p\ papID\ uID')$

**and**

$\bigwedge s\ confID\ uID\ p\ papID\ uID'.$

$\llbracket reach\ s; \varphi\ s; e\ createReview\ s\ confID\ uID\ p\ papID\ uID' \rrbracket$   
 $\implies \varphi\ (createReview\ s\ confID\ uID\ p\ papID\ uID')$

**shows**  $cIsInvar\ \varphi$

**unfolding**  $cIsInvar\ def$  **apply**  $safe$  **subgoal for - ca using**  $assms$  **by** ( $cases\ ca,$

*auto*) .

**lemma**  $uIsInvar[case-names\ uUser\ uConfA\ uNextPhase\ uPaperTA\ uPaperC\ uPref\ uReview]$ :

**assumes**

$\bigwedge s\ uID\ p\ p'\ name\ info\ email.$

$\llbracket reach\ s; \varphi\ s; e-updateUser\ s\ uID\ p\ p'\ name\ info\ email \rrbracket$

$\implies \varphi\ (updateUser\ s\ uID\ p\ p'\ name\ info\ email)$

**and**

$\bigwedge s\ confID\ uID\ p.$

$\llbracket reach\ s; \varphi\ s; e-updateConfA\ s\ confID\ uID\ p \rrbracket \implies \varphi\ (updateConfA\ s\ confID$

$uID\ p)$

**and**

$\bigwedge s\ confID\ uID\ p\ ph.$

$\llbracket reach\ s; \varphi\ s; e-updatePhase\ s\ confID\ uID\ p\ ph \rrbracket \implies \varphi\ (updatePhase\ s\ confID$

$uID\ p\ ph)$

**and**

$\bigwedge s\ confID\ uID\ p\ paperID\ name\ info.$

$\llbracket reach\ s; \varphi\ s; e-updatePaperTA\ s\ confID\ uID\ p\ paperID\ name\ info \rrbracket$

$\implies \varphi\ (updatePaperTA\ s\ confID\ uID\ p\ paperID\ name\ info)$

**and**

$\bigwedge s\ confID\ uID\ p\ paperID\ pc.$

$\llbracket reach\ s; \varphi\ s; e-updatePaperC\ s\ confID\ uID\ p\ paperID\ pc \rrbracket$

$\implies \varphi\ (updatePaperC\ s\ confID\ uID\ p\ paperID\ pc)$

**and**

$\bigwedge s\ confID\ uID\ p\ paperID\ preference.$

$\llbracket reach\ s; \varphi\ s; e-updatePref\ s\ confID\ uID\ p\ paperID\ preference \rrbracket$

$\implies \varphi\ (updatePref\ s\ confID\ uID\ p\ paperID\ preference)$

**and**

$\bigwedge s\ confID\ uID\ p\ paperID\ n\ rc.$

$\llbracket reach\ s; \varphi\ s; e-updateReview\ s\ confID\ uID\ p\ paperID\ n\ rc \rrbracket$

$\implies \varphi\ (updateReview\ s\ confID\ uID\ p\ paperID\ n\ rc)$

**and**

$\bigwedge s\ confID\ uID\ p\ paperID\ fpc.$

$\llbracket reach\ s; \varphi\ s; e-updateFPaperC\ s\ confID\ uID\ p\ paperID\ fpc \rrbracket$

$\implies \varphi\ (updateFPaperC\ s\ confID\ uID\ p\ paperID\ fpc)$

**shows**  $uIsInvar\ \varphi$

**unfolding**  $uIsInvar-def$  **apply safe using**  $assms$  **subgoal for** -  $ua$  **by** ( $cases\ ua,$   
 $auto$ ) .

**lemma**  $uuIsInvar[case-names\ uuNews\ uuDis\ uuReview\ uuDec]$ :

**assumes**

$\bigwedge s\ confID\ uID\ p\ comm.$

$\llbracket reach\ s; \varphi\ s; e-uupdateNews\ s\ confID\ uID\ p\ comm \rrbracket$

$\implies \varphi\ (uupdateNews\ s\ confID\ uID\ p\ comm)$

**and**

$\bigwedge s\ confID\ uID\ p\ paperID\ comm.$

$\llbracket reach\ s; \varphi\ s; e-uupdateDis\ s\ confID\ uID\ p\ paperID\ comm \rrbracket$

$\implies \varphi\ (uupdateDis\ s\ confID\ uID\ p\ paperID\ comm)$

**and**  
 $\wedge s \text{ confID } uID \text{ } p \text{ } paperID \text{ } n \text{ } rc.$   
 $\llbracket \text{reach } s; \varphi \text{ } s; e\text{-uupdateReview } s \text{ } confID \text{ } uID \text{ } p \text{ } paperID \text{ } n \text{ } rc \rrbracket$   
 $\implies \varphi (\text{uupdateReview } s \text{ } confID \text{ } uID \text{ } p \text{ } paperID \text{ } n \text{ } rc)$   
**and**  
 $\wedge s \text{ confID } uID \text{ } p \text{ } paperID \text{ } decision.$   
 $\llbracket \text{reach } s; \varphi \text{ } s; e\text{-uupdateDec } s \text{ } confID \text{ } uID \text{ } p \text{ } paperID \text{ } decision \rrbracket$   
 $\implies \varphi (\text{uupdateDec } s \text{ } confID \text{ } uID \text{ } p \text{ } paperID \text{ } decision)$   
**shows**  $uIsInvar \text{ } \varphi$   
**unfolding**  $uIsInvar\text{-def}$  **apply** *safe subgoal for - uua using assms by (cases uua, auto)* .

### 3.2 Safety proofs

**declare**  $option.\text{splits}[split] \text{ } paper.\text{splits}[split] \text{ } discussion.\text{splits}[split] \text{ } role.\text{splits}[split]$   
 $Let\text{-def}[simp] \text{ } list\text{-all}\text{-iff}[simp] \text{ } list\text{-ex}\text{-iff}[simp] \text{ } fun\text{-upd2}\text{-def}[simp] \text{ } IDsOK\text{-def}[simp]$   
 $if\text{-splits}[split]$

**fun**  $papIDsOfRole$  **where**  
 $papIDsOfRole \text{ } (Aut \text{ } papID) = [papID]$   
 $|$   
 $papIDsOfRole \text{ } (Rev \text{ } papID \text{ } n) = [papID]$   
 $|$   
 $papIDsOfRole \text{ } - = []$

**definition**  $phase\text{-leq}\text{-closedPH} :: state \Rightarrow bool$  **where**  
 $phase\text{-leq}\text{-closedPH} \text{ } s \equiv$   
 $\forall \text{ confID. } phase \text{ } s \text{ } confID \leq \text{ } closedPH$

**lemma**  $holdsIstate\text{-phase}\text{-leq}\text{-closedPH}$ :  $holdsIstate \text{ } phase\text{-leq}\text{-closedPH}$   
**unfolding**  $IO\text{-Automaton.holdsIstate}\text{-def}$   $istate\text{-def}$   $phase\text{-leq}\text{-closedPH}\text{-def}$  **by** *auto*

**lemma**  $cIsInvar\text{-phase}\text{-leq}\text{-closedPH}$ :  $cIsInvar \text{ } phase\text{-leq}\text{-closedPH}$   
**apply** (*cases phase-leq-closedPH rule: cIsInvar*)  
**by** (*auto simp: c-defs phase-leq-closedPH-def*)

**lemma**  $uIsInvar\text{-phase}\text{-leq}\text{-closedPH}$ :  $uIsInvar \text{ } phase\text{-leq}\text{-closedPH}$   
**apply** (*cases phase-leq-closedPH rule: uIsInvar*)  
**by** (*auto simp: u-defs phase-leq-closedPH-def*)

**lemma**  $uuIsInvar\text{-phase}\text{-leq}\text{-closedPH}$ :  $uuIsInvar \text{ } phase\text{-leq}\text{-closedPH}$   
**apply** (*cases phase-leq-closedPH rule: uuIsInvar*)  
**by** (*auto simp: uu-defs phase-leq-closedPH-def*)

**lemma**  $invar\text{-phase}\text{-leq}\text{-closedPH}$ :  $invar \text{ } phase\text{-leq}\text{-closedPH}$   
**unfolding**  $invar\text{-cIsInvar}\text{-uIsInvar}\text{-uuIsInvar}$   
**using**  $cIsInvar\text{-phase}\text{-leq}\text{-closedPH}$   $uIsInvar\text{-phase}\text{-leq}\text{-closedPH}$   $uuIsInvar\text{-phase}\text{-leq}\text{-closedPH}$

**by** *auto*

**lemmas** *phase-leq-closedPH1* =  
*holdsIstate-invar*[*OF holdsIstate-phase-leq-closedPH invar-phase-leq-closedPH*]

**theorem** *phase-leq-closedPH*:  
**assumes** *a*: *reach s*  
**shows** *phase s confID*  $\leq$  *closedPH*  
**using** *phase-leq-closedPH1*[*OF a*] **unfolding** *phase-leq-closedPH-def* **by** *auto*

**definition** *geq-noPH-confIDs* :: *state*  $\Rightarrow$  *bool* **where**  
*geq-noPH-confIDs s*  $\equiv$   
 $\forall$  *confID*. *phase s confID*  $>$  *noPH*  $\longrightarrow$  *confID*  $\in\in$  *confIDs s*

**lemma** *holdsIstate-geq-noPH-confIDs*: *holdsIstate geq-noPH-confIDs*  
**unfolding** *IO-Automaton.holdsIstate-def istate-def istate-def geq-noPH-confIDs-def*  
**by** *auto*

**lemma** *cIsInvar-geq-noPH-confIDs*: *cIsInvar geq-noPH-confIDs*  
**apply** (*cases geq-noPH-confIDs rule: cIsInvar*)  
**by** (*auto simp: c-defs geq-noPH-confIDs-def*)

**lemma** *uIsInvar-geq-noPH-confIDs*: *uIsInvar geq-noPH-confIDs*  
**apply** (*cases geq-noPH-confIDs rule: uIsInvar*)  
**by** (*auto simp: u-defs geq-noPH-confIDs-def*)

**lemma** *uuIsInvar-geq-noPH-confIDs*: *uuIsInvar geq-noPH-confIDs*  
**apply** (*cases geq-noPH-confIDs rule: uuIsInvar*)  
**by** (*auto simp: uu-defs geq-noPH-confIDs-def*)

**lemma** *invar-geq-noPH-confIDs*: *invar geq-noPH-confIDs*  
**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*  
**using** *cIsInvar-geq-noPH-confIDs uIsInvar-geq-noPH-confIDs uuIsInvar-geq-noPH-confIDs*  
**by** *auto*

**lemmas** *geq-noPH-confIDs1* =  
*holdsIstate-invar*[*OF holdsIstate-geq-noPH-confIDs invar-geq-noPH-confIDs*]

**theorem** *geq-noPH-confIDs*:  
**assumes** *a*: *reach s*  
**shows** *phase s confID*  $>$  *noPH*  $\longrightarrow$  *confID*  $\in\in$  *confIDs s*  
**using** *geq-noPH-confIDs1*[*OF a*] **unfolding** *geq-noPH-confIDs-def* **by** *auto*

**definition** *roles-IDsOK* :: *state*  $\Rightarrow$  *bool* **where**  
*roles-IDsOK s*  $\equiv$   
 $\forall$  *confID uID rl*.  
 $rl \in\in$  *roles s confID uID*  $\longrightarrow$  *IDsOK s* [*confID*] [*uID*] (*papIDsOfRole rl*)

**lemma** *holdsIstate-roles-IDsOK*: *holdsIstate roles-IDsOK*  
**unfolding** *IO-Automaton.holdsIstate-def istate-def istate-def roles-IDsOK-def* **by**  
*auto*

**lemma** *cIsInvar-roles-IDsOK*: *cIsInvar roles-IDsOK*  
**apply** (*cases roles-IDsOK rule: cIsInvar*)  
**by** (*auto simp: c-defs roles-IDsOK-def*)

**lemma** *uIsInvar-roles-IDsOK*: *uIsInvar roles-IDsOK*  
**apply** (*cases roles-IDsOK rule: uIsInvar*)  
**by** (*auto simp: u-defs roles-IDsOK-def*)

**lemma** *uuIsInvar-roles-IDsOK*: *uuIsInvar roles-IDsOK*  
**apply** (*cases roles-IDsOK rule: uuIsInvar*)  
**by** (*auto simp: uu-defs roles-IDsOK-def*)

**lemma** *invar-roles-IDsOK*: *invar roles-IDsOK*  
**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*  
**using** *cIsInvar-roles-IDsOK uIsInvar-roles-IDsOK uuIsInvar-roles-IDsOK* **by** *auto*

**lemmas** *roles-IDsOK1* =  
*holdsIstate-invar[OF holdsIstate-roles-IDsOK invar-roles-IDsOK]*

**theorem** *roles-IDsOK*:  
**assumes** *a: reach s* **and** *rl: rl ∈∈ roles s confID uID*  
**shows** *IDsOK s [confID] [uID] (papIDsOfRole rl)*  
**using** *roles-IDsOK1[OF a] rl* **unfolding** *roles-IDsOK-def* **by** *auto*

**corollary** *roles-confIDs*:  
**assumes** *a: reach s* **and** *A: rl ∈∈ roles s confID uID*  
**shows** *confID ∈∈ confIDs s*  
**using** *roles-IDsOK[OF a] A* **unfolding** *IDsOK-def* **by** *auto*

**corollary** *roles-userIDs*:  
**assumes** *a: reach s* **and** *A: rl ∈∈ roles s confID uID*  
**shows** *uID ∈∈ userIDs s*  
**using** *roles-IDsOK[OF a] A* **unfolding** *IDsOK-def* **by** *auto*

**corollary** *isAut-paperIDs*:  
**assumes** *a: reach s* **and** *A: isAut s confID uID papID*  
**shows** *papID ∈∈ paperIDs s confID*  
**using** *roles-IDsOK[OF a] A* **unfolding** *IDsOK-def* **by** *auto*

**corollary** *isRevNth-paperIDs*:  
**assumes** *a: reach s* **and** *A: isRevNth s confID uID papID n*  
**shows** *papID ∈∈ paperIDs s confID*  
**using** *roles-IDsOK[OF a] A* **unfolding** *IDsOK-def* **by** *auto*

**corollary** *isRev-paperIDs*:  
**assumes**  $a$ : reach  $s$  **and**  $A$ : *isRev*  $s$  *confID*  $uID$   $papID$   
**shows**  $papID \in \in paperIDs\ s\ confID$   
**using** *isRevNth-paperIDs*[*OF a*]  $A$  **unfolding** *isRev-def2* **by** *auto*

**corollary** *isRev-userIDs*:  
**assumes**  $a$ : reach  $s$  **and**  $A$ : *isRev*  $s$  *confID*  $uID$   $papID$   
**shows**  $uID \in \in userIDs\ s$   
**using** *roles-userIDs*[*OF a*]  $A$  **unfolding** *isRev-def2* **by** *auto*

**corollary** *isRev-confIDs*:  
**assumes**  $a$ : reach  $s$  **and**  $A$ : *isRev*  $s$  *confID*  $uID$   $papID$   
**shows**  $confID \in \in confIDs\ s$   
**using** *roles-confIDs*[*OF a*]  $A$  **unfolding** *isRev-def2* **by** *auto*

**definition** *distinct-IDs* :: state  $\Rightarrow$  bool **where**  
*distinct-IDs*  $s \equiv$   
 $distinct\ (confIDs\ s) \wedge distinct\ (userIDs\ s) \wedge (\forall\ confID.\ distinct\ (paperIDs\ s\ confID))$

**lemma** *holdsIstate-distinct-IDs*: *holdsIstate* *distinct-IDs*  
**unfolding** *IO-Automaton.holdsIstate-def* *istate-def* *istate-def* *distinct-IDs-def* **by** *auto*

**lemma** *cIsInvar-distinct-IDs*: *cIsInvar* *distinct-IDs*  
**apply** (*cases* *distinct-IDs* *rule*: *cIsInvar*)  
**by** (*auto simp*: *c-defs* *distinct-IDs-def* *getAllPaperIDs-def*)

**lemma** *uIsInvar-distinct-IDs*: *uIsInvar* *distinct-IDs*  
**apply** (*cases* *distinct-IDs* *rule*: *uIsInvar*)  
**by** (*auto simp*: *u-defs* *distinct-IDs-def*)

**lemma** *uuIsInvar-distinct-IDs*: *uuIsInvar* *distinct-IDs*  
**apply** (*cases* *distinct-IDs* *rule*: *uuIsInvar*)  
**by** (*auto simp*: *uu-defs* *distinct-IDs-def*)

**lemma** *invar-distinct-IDs*: *invar* *distinct-IDs*  
**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*  
**using** *cIsInvar-distinct-IDs* *uIsInvar-distinct-IDs* *uuIsInvar-distinct-IDs* **by** *auto*

**lemmas** *distinct-IDs1* = *holdsIstate-invar*[*OF holdsIstate-distinct-IDs invar-distinct-IDs*]

**theorem** *distinct-IDs*:  
**assumes**  $a$ : reach  $s$   
**shows**  $distinct\ (confIDs\ s) \wedge distinct\ (userIDs\ s) \wedge (\forall\ confID.\ distinct\ (paperIDs\ s\ confID))$   
**using** *distinct-IDs1*[*OF a*] **unfolding** *distinct-IDs-def* **by** *auto*

**lemmas** *distinct-confIDs* = *distinct-IDs*[*THEN conjunct1*]  
**lemmas** *distinct-userIDs* = *distinct-IDs*[*THEN conjunct2*, *THEN conjunct1*]  
**lemmas** *distinct-paperIDs* = *distinct-IDs*[*THEN conjunct2*, *THEN conjunct2*, *rule-format*]

**definition** *distinct-roles* :: *state*  $\Rightarrow$  *bool* **where**  
*distinct-roles* *s*  $\equiv$   
 $\forall$  *confID uID*. *distinct* (*roles s confID uID*)

**lemma** *holdsIstate-distinct-roles*: *holdsIstate distinct-roles*  
**unfolding** *IO-Automaton.holdsIstate-def istate-def istate-def distinct-roles-def* **by**  
*auto*

**lemma** *cIsInvar-distinct-roles*: *cIsInvar distinct-roles*  
**apply** (*cases distinct-roles rule: cIsInvar*)  
**by** (*auto simp: c-defs distinct-roles-def*)

**lemma** *uIsInvar-distinct-roles*: *uIsInvar distinct-roles*  
**apply** (*cases distinct-roles rule: uIsInvar*)  
**by** (*auto simp: u-defs distinct-roles-def*)

**lemma** *uuIsInvar-distinct-roles*: *uuIsInvar distinct-roles*  
**apply** (*cases distinct-roles rule: uuIsInvar*)  
**by** (*auto simp: uu-defs distinct-roles-def*)

**lemma** *invar-distinct-roles*: *invar distinct-roles*  
**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*  
**using** *cIsInvar-distinct-roles uIsInvar-distinct-roles uuIsInvar-distinct-roles* **by** *auto*

**lemmas** *distinct-roles1* = *holdsIstate-invar*[*OF holdsIstate-distinct-roles invar-distinct-roles*]

**theorem** *distinct-roles*:  
**assumes** *a*: *reach s*  
**shows** *distinct* (*roles s confID uID*)  
**using** *distinct-roles1*[*OF a*] **unfolding** *distinct-roles-def* **by** *auto*

**definition** *isRevNth-isPC* :: *state*  $\Rightarrow$  *bool* **where**  
*isRevNth-isPC* *s*  $\equiv$   
 $\forall$  *confID uID papID n*. *isRevNth s confID uID papID n*  $\longrightarrow$  *isPC s confID uID*

**lemma** *holdsIstate-isRevNth-isPC*: *holdsIstate isRevNth-isPC*  
**unfolding** *IO-Automaton.holdsIstate-def istate-def istate-def isRevNth-isPC-def* **by**  
*auto*

**lemma** *cIsInvar-isRevNth-isPC*: *cIsInvar isRevNth-isPC*  
**apply** (*cases isRevNth-isPC rule: cIsInvar*)  
**by** (*auto simp: c-defs isRevNth-isPC-def*)

**lemma** *uIsInvar-isRevNth-isPC*: *uIsInvar isRevNth-isPC*  
**apply** (*cases isRevNth-isPC rule: uIsInvar*)  
**by** (*auto simp: u-defs isRevNth-isPC-def*)

**lemma** *uuIsInvar-isRevNth-isPC*: *uuIsInvar isRevNth-isPC*  
**apply** (*cases isRevNth-isPC rule: uuIsInvar*)  
**by** (*auto simp: uu-defs isRevNth-isPC-def*)

**lemma** *invar-isRevNth-isPC*: *invar isRevNth-isPC*  
**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*  
**using** *cIsInvar-isRevNth-isPC uIsInvar-isRevNth-isPC uuIsInvar-isRevNth-isPC* **by**  
*auto*

**lemmas** *isRevNth-isPC1 = holdsIstate-invar[OF holdsIstate-isRevNth-isPC invar-isRevNth-isPC]*

**theorem** *isRevNth-isPC*:  
**assumes** *a: reach s* **and** *R: isRevNth s confID uID papID n*  
**shows** *isPC s confID uID*  
**using** *isRevNth-isPC1[OF a] R* **unfolding** *isRevNth-isPC-def* **by** *auto*

**corollary** *isRev-isPC*:  
**assumes** *a: reach s* **and** *R: isRev s confID uID papID*  
**shows** *isPC s confID uID*  
**using** *isRevNth-isPC[OF a] R* **unfolding** *isRev-def2* **by** *auto*

**definition** *paperIDs-confIDs* :: *state*  $\Rightarrow$  *bool* **where**  
*paperIDs-confIDs s*  $\equiv$   
 $\forall$  *confID papID*.  
*papID*  $\in\in$  *paperIDs s confID*  $\longrightarrow$  *confID*  $\in\in$  *confIDs s*

**lemma** *holdsIstate-paperIDs-confIDs*: *holdsIstate paperIDs-confIDs*  
**unfolding** *IO-Automaton.holdsIstate-def istate-def istate-def paperIDs-confIDs-def*  
**by** *auto*

**lemma** *cIsInvar-paperIDs-confIDs*: *cIsInvar paperIDs-confIDs*  
**apply** (*cases paperIDs-confIDs rule: cIsInvar*)  
**by** (*auto simp: c-defs paperIDs-confIDs-def*)

**lemma** *uIsInvar-paperIDs-confIDs*: *uIsInvar paperIDs-confIDs*  
**apply** (*cases paperIDs-confIDs rule: uIsInvar*)  
**by** (*auto simp: u-defs paperIDs-confIDs-def*)

**lemma** *uuIsInvar-paperIDs-confIDs*: *uuIsInvar paperIDs-confIDs*  
**apply** (*cases paperIDs-confIDs rule: uuIsInvar*)  
**by** (*auto simp: uu-defs paperIDs-confIDs-def*)

**lemma** *invar-paperIDs-confIDs*: *invar paperIDs-confIDs*  
**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*

**using** *cIsInvar-paperIDs-confIDs* *uIsInvar-paperIDs-confIDs* *uuIsInvar-paperIDs-confIDs*  
**by** *auto*

**lemmas** *paperIDs-confIDs1* = *holdsIstate-invar*[*OF holdsIstate-paperIDs-confIDs*  
*invar-paperIDs-confIDs*]

**theorem** *paperIDs-confIDs*:

**assumes** *a*: *reach s* **and** *p*: *papID*  $\in\in$  *paperIDs s confID*

**shows** *confID*  $\in\in$  *confIDs s*

**using** *paperIDs-confIDs1*[*OF a*] *p* **unfolding** *paperIDs-confIDs-def* **by** *auto*

**corollary** *paperIDs-getAllPaperIDs*:

**assumes** *a*: *reach s* **and** *p*: *papID*  $\in\in$  *paperIDs s confID*

**shows** *papID*  $\in\in$  *getAllPaperIDs s*

**using** *paperIDs-confIDs*[*OF assms*] *p* **unfolding** *getAllPaperIDs-def* **by** *auto*

**corollary** *isRevNth-getAllPaperIDs*:

**assumes** *a*: *reach s* **and** *isRevNth s confID uID papID n*

**shows** *papID*  $\in\in$  *getAllPaperIDs s*

**using** *paperIDs-getAllPaperIDs*[*OF a isRevNth-paperIDs*[*OF assms*]] .

**definition** *paperIDs-equals* :: *state*  $\Rightarrow$  *bool* **where**

*paperIDs-equals s*  $\equiv$

$\forall$  *confID1 confID2 papID*.

*papID*  $\in\in$  *paperIDs s confID1*  $\wedge$  *papID*  $\in\in$  *paperIDs s confID2*

$\longrightarrow$  *confID1* = *confID2*

**lemma** *holdsIstate-paperIDs-equals*: *holdsIstate paperIDs-equals*

**unfolding** *IO-Automaton.holdsIstate-def* *istate-def* *istate-def paperIDs-equals-def*

**by** *auto*

**lemma** *cIsInvar-paperIDs-equals*: *cIsInvar paperIDs-equals*

**apply** (*cases paperIDs-equals rule: cIsInvar*)

**by** (*auto simp: c-defs paperIDs-equals-def paperIDs-getAllPaperIDs*)

**lemma** *uIsInvar-paperIDs-equals*: *uIsInvar paperIDs-equals*

**apply** (*cases paperIDs-equals rule: uIsInvar*)

**by** (*auto simp: u-defs paperIDs-equals-def*)

**lemma** *uuIsInvar-paperIDs-equals*: *uuIsInvar paperIDs-equals*

**apply** (*cases paperIDs-equals rule: uuIsInvar*)

**by** (*auto simp: uu-defs paperIDs-equals-def*)

**lemma** *invar-paperIDs-equals*: *invar paperIDs-equals*

**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*

**using** *cIsInvar-paperIDs-equals* *uIsInvar-paperIDs-equals* *uuIsInvar-paperIDs-equals*

**by** *auto*

**lemmas** *paperIDs-equals1* = *holdsIstate-invar*[*OF holdsIstate-paperIDs-equals invar-paperIDs-equals*]

**theorem** *paperIDs-equals*:

**assumes** *a*: *reach s* **and** *p*: *papID*  $\in\in$  *paperIDs s* *confID1 papID*  $\in\in$  *paperIDs s* *confID2*

**shows** *confID1 = confID2*

**using** *paperIDs-equals1*[*OF a*] *p* **unfolding** *paperIDs-equals-def* **by** *auto*

**definition** *isAut-pref-Conflict* :: *state*  $\Rightarrow$  *bool* **where**

*isAut-pref-Conflict s*  $\equiv$

$\forall$  *confID uID papID*. *isAut s confID uID papID*  $\longrightarrow$  *pref s uID papID = Conflict*

**lemma** *holdsIstate-isAut-pref-Conflict*: *holdsIstate isAut-pref-Conflict*

**unfolding** *IO-Automaton.holdsIstate-def istate-def istate-def isAut-pref-Conflict-def* **by** *auto*

**lemma** *cIsInvar-isAut-pref-Conflict*: *cIsInvar isAut-pref-Conflict*

**apply** (*cases isAut-pref-Conflict* *rule: cIsInvar*)

**by** (*auto simp: c-defs isAut-pref-Conflict-def*)

**lemma** *uIsInvar-isAut-pref-Conflict*: *uIsInvar isAut-pref-Conflict*

**proof** (*cases isAut-pref-Conflict* *rule: uIsInvar*)

**case** (*uPref s confID uID p paperID preference*)

**thus** ?*case* **apply** (*auto simp: u-defs isAut-pref-Conflict-def*)

**apply** (*frule isAut-paperIDs, simp*)

**apply** (*frule paperIDs-equals, simp, simp, fastforce*)

**done**

**qed** (*auto simp: u-defs isAut-pref-Conflict-def*)

**lemma** *uuIsInvar-isAut-pref-Conflict*: *uuIsInvar isAut-pref-Conflict*

**apply** (*cases isAut-pref-Conflict* *rule: uuIsInvar*)

**by** (*auto simp: uu-defs isAut-pref-Conflict-def*)

**lemma** *invar-isAut-pref-Conflict*: *invar isAut-pref-Conflict*

**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*

**using** *cIsInvar-isAut-pref-Conflict uIsInvar-isAut-pref-Conflict*

*uuIsInvar-isAut-pref-Conflict* **by** *auto*

**lemmas** *isAut-pref-Conflict1* =

*holdsIstate-invar*[*OF holdsIstate-isAut-pref-Conflict invar-isAut-pref-Conflict*]

**theorem** *isAut-pref-Conflict*:

**assumes** *a*: *reach s* **and** *i*: *isAut s confID uID papID*

**shows** *pref s uID papID = Conflict*

**using** *isAut-pref-Conflict1*[*OF a*] *i* **unfolding** *isAut-pref-Conflict-def* **by** *auto*

**definition** *phase-noPH-paperIDs* :: state  $\Rightarrow$  bool **where**

*phase-noPH-paperIDs* s  $\equiv$

$\forall$  confID. phase s confID = noPH  $\longrightarrow$  paperIDs s confID = []

**lemma** *holdsIstate-phase-noPH-paperIDs*: holdsIstate *phase-noPH-paperIDs*

**unfolding** *IO-Automaton.holdsIstate-def istate-def istate-def phase-noPH-paperIDs-def*  
**by** *auto*

**lemma** *cIsInvar-phase-noPH-paperIDs*: *cIsInvar phase-noPH-paperIDs*

**apply** (*cases phase-noPH-paperIDs rule: cIsInvar*)

**by** (*auto simp: c-defs phase-noPH-paperIDs-def*)

**lemma** *uIsInvar-phase-noPH-paperIDs*: *uIsInvar phase-noPH-paperIDs*

**apply** (*cases phase-noPH-paperIDs rule: uIsInvar*)

**by** (*auto simp: u-defs phase-noPH-paperIDs-def*)

**lemma** *uuIsInvar-phase-noPH-paperIDs*: *uuIsInvar phase-noPH-paperIDs*

**apply** (*cases phase-noPH-paperIDs rule: uuIsInvar*)

**by** (*auto simp: uu-defs phase-noPH-paperIDs-def*)

**lemma** *invar-phase-noPH-paperIDs*: *invar phase-noPH-paperIDs*

**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*

**using** *cIsInvar-phase-noPH-paperIDs uIsInvar-phase-noPH-paperIDs*

*uuIsInvar-phase-noPH-paperIDs* **by** *auto*

**lemmas** *phase-noPH-paperIDs1* =

*holdsIstate-invar[OF holdsIstate-phase-noPH-paperIDs invar-phase-noPH-paperIDs]*

**theorem** *phase-noPH-paperIDs*:

**assumes** *a*: reach s **and** *p*: phase s confID = noPH

**shows** paperIDs s confID = []

**using** *phase-noPH-paperIDs1[OF a] p* **unfolding** *phase-noPH-paperIDs-def* **by**  
*auto*

**definition** *paperIDs-geq-subPH* :: state  $\Rightarrow$  bool **where**

*paperIDs-geq-subPH* s  $\equiv$

$\forall$  confID papID. papID  $\in$  paperIDs s confID  $\longrightarrow$  phase s confID  $\geq$  subPH

**lemma** *holdsIstate-paperIDs-geq-subPH*: holdsIstate *paperIDs-geq-subPH*

**unfolding** *IO-Automaton.holdsIstate-def istate-def istate-def paperIDs-geq-subPH-def*  
**by** *auto*

**lemma** *cIsInvar-paperIDs-geq-subPH*: *cIsInvar paperIDs-geq-subPH*

**apply** (*cases paperIDs-geq-subPH rule: cIsInvar*)

**by** (*auto simp: c-defs paperIDs-geq-subPH-def*)

**lemma** *uIsInvar-paperIDs-geq-subPH*: *uIsInvar paperIDs-geq-subPH*

**apply** (*cases paperIDs-geq-subPH rule: uIsInvar*)

**by** (*fastforce simp: u-defs paperIDs-geq-subPH-def*)**+**

**lemma** *uuIsInvar-paperIDs-geq-subPH*: *uuIsInvar paperIDs-geq-subPH*  
**apply** (*cases paperIDs-geq-subPH rule: uuIsInvar*)  
**by** (*auto simp: uu-defs paperIDs-geq-subPH-def*)

**lemma** *invar-paperIDs-geq-subPH*: *invar paperIDs-geq-subPH*  
**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*  
**using** *cIsInvar-paperIDs-geq-subPH uIsInvar-paperIDs-geq-subPH*  
*uuIsInvar-paperIDs-geq-subPH* **by** *auto*

**lemmas** *paperIDs-geq-subPH1* =  
*holdsIstate-invar[OF holdsIstate-paperIDs-geq-subPH invar-paperIDs-geq-subPH]*

**theorem** *paperIDs-geq-subPH*:  
**assumes** *a: reach s* **and** *i: papID ∈∈ paperIDs s confID*  
**shows** *phase s confID ≥ subPH*  
**using** *paperIDs-geq-subPH1[OF a] i unfolding paperIDs-geq-subPH-def* **by** *auto*

**definition** *isRevNth-geq-revPH* :: *state ⇒ bool* **where**  
*isRevNth-geq-revPH s ≡*  
*∀ confID uID papID n. isRevNth s confID uID papID n → phase s confID ≥*  
*revPH*

**lemma** *holdsIstate-isRevNth-geq-revPH*: *holdsIstate isRevNth-geq-revPH*  
**unfolding** *IO-Automaton.holdsIstate-def istate-def istate-def isRevNth-geq-revPH-def*  
**by** *auto*

**lemma** *cIsInvar-isRevNth-geq-revPH*: *cIsInvar isRevNth-geq-revPH*  
**apply** (*cases isRevNth-geq-revPH rule: cIsInvar*)  
**by** (*auto simp: c-defs isRevNth-geq-revPH-def*)

**lemma** *uIsInvar-isRevNth-geq-revPH*: *uIsInvar isRevNth-geq-revPH*  
**proof** (*cases isRevNth-geq-revPH rule: uIsInvar*)  
  **case** (*uConfA s confID uID p*) **thus** ?*case*  
  **by** (*fastforce simp: u-defs isRevNth-geq-revPH-def*)  
**qed**(*fastforce simp: u-defs isRevNth-geq-revPH-def*)**+**

**lemma** *uuIsInvar-isRevNth-geq-revPH*: *uuIsInvar isRevNth-geq-revPH*  
**apply** (*cases isRevNth-geq-revPH rule: uuIsInvar*)  
**by** (*auto simp: uu-defs isRevNth-geq-revPH-def*)

**lemma** *invar-isRevNth-geq-revPH*: *invar isRevNth-geq-revPH*  
**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*  
**using** *cIsInvar-isRevNth-geq-revPH uIsInvar-isRevNth-geq-revPH*  
*uuIsInvar-isRevNth-geq-revPH* **by** *auto*

**lemmas** *isRevNth-geq-revPH1* =

*holdsIstate-invar*[*OF holdsIstate-isRevNth-geq-revPH invar-isRevNth-geq-revPH*]

**theorem** *isRevNth-geq-revPH*:

**assumes** *a*: *reach s* **and** *i*: *isRevNth s confID uID papID n*

**shows** *phase s confID*  $\geq$  *revPH*

**using** *isRevNth-geq-revPH1*[*OF a*] *i* **unfolding** *isRevNth-geq-revPH-def* **by** *auto*

**corollary** *isRev-geq-revPH*:

**assumes** *a*: *reach s* **and** *i*: *isRev s confID uID papID*

**shows** *phase s confID*  $\geq$  *revPH*

**using** *isRevNth-geq-revPH*[*OF a*] *i* **unfolding** *isRev-def2* **by** *auto*

**definition** *paperID-ex-userID* :: *state*  $\Rightarrow$  *bool* **where**

*paperID-ex-userID s*  $\equiv$

$\forall$  *confID papID*. *papID*  $\in$  *paperIDs s confID*  $\longrightarrow$  ( $\exists$  *uID*. *isAut s confID uID papID*)

**lemma** *holdsIstate-paperID-ex-userID*: *holdsIstate paperID-ex-userID*

**unfolding** *IO-Automaton.holdsIstate-def istate-def istate-def paperID-ex-userID-def*  
**by** *auto*

**lemma** *cIsInvar-paperID-ex-userID*: *cIsInvar paperID-ex-userID*

**apply** (*cases paperID-ex-userID rule: cIsInvar*)

**by** (*fastforce simp: c-defs paperID-ex-userID-def paperIDs-confIDs*) $+$

**lemma** *uIsInvar-paperID-ex-userID*: *uIsInvar paperID-ex-userID*

**apply** (*cases paperID-ex-userID rule: uIsInvar*)

**by** (*fastforce simp: u-defs paperID-ex-userID-def*) $+$

**lemma** *uuIsInvar-paperID-ex-userID*: *uuIsInvar paperID-ex-userID*

**apply** (*cases paperID-ex-userID rule: uuIsInvar*)

**by** (*auto simp: uu-defs paperID-ex-userID-def*)

**lemma** *invar-paperID-ex-userID*: *invar paperID-ex-userID*

**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*

**using** *cIsInvar-paperID-ex-userID uIsInvar-paperID-ex-userID*

*uuIsInvar-paperID-ex-userID* **by** *auto*

**lemmas** *paperID-ex-userID1* =

*holdsIstate-invar*[*OF holdsIstate-paperID-ex-userID invar-paperID-ex-userID*]

**theorem** *paperID-ex-userID*:

**assumes** *a*: *reach s* **and** *i*: *papID*  $\in$  *paperIDs s confID*

**shows**  $\exists$  *uID*. *isAut s confID uID papID*

**using** *paperID-ex-userID1*[*OF a*] *i* **unfolding** *paperID-ex-userID-def* **by** *auto*

**definition** *pref-Conflict-isRevNth* :: *state*  $\Rightarrow$  *bool* **where**

*pref-Conflict-isRevNth*  $s \equiv$   
 $\forall \text{ confID } uID \text{ papID } n. \text{ pref } s \text{ } uID \text{ } papID = \text{ Conflict} \longrightarrow \neg \text{ isRevNth } s \text{ } \text{ confID } uID$   
*papID*  $n$

**lemma** *holdsIstate-pref-Conflict-isRevNth*: *holdsIstate* *pref-Conflict-isRevNth*  
**unfolding** *IO-Automaton.holdsIstate-def* *istate-def* *istate-def* *pref-Conflict-isRevNth-def*  
**by** *auto*

**lemma** *cIsInvar-pref-Conflict-isRevNth*: *cIsInvar* *pref-Conflict-isRevNth*

**proof** (*cases* *pref-Conflict-isRevNth* *rule*: *cIsInvar*)  
**case** (*cAuthor*  $s \text{ } \text{ confID } uID \text{ } p \text{ } \text{ papID } uID'$ ) **thus** ?*case*  
**apply** (*auto simp*: *c-defs* *pref-Conflict-isRevNth-def*)  
**apply** (*frule* *isRevNth-geq-revPH*, *simp*, *simp*)  
**apply** (*frule* *isRevNth-paperIDs*, *simp*)  
**apply** (*frule* *paperIDs-equals*, *simp*, *simp*, *force*)  
**done**

**next**

**case** (*cConflict*  $s \text{ } \text{ confID } uID \text{ } p \text{ } \text{ papID } uID'$ ) **thus** ?*case*  
**apply** (*auto simp*: *c-defs* *pref-Conflict-isRevNth-def*)  
**apply** (*frule* *isRevNth-geq-revPH*, *simp*)  
**apply** (*frule* *isRevNth-paperIDs*, *simp*)  
**apply** (*frule* *paperIDs-equals*, *simp*, *simp*, *force*)  
**done**

**qed** (*auto simp*: *c-defs* *pref-Conflict-isRevNth-def* *isRevNth-getAllPaperIDs*)

**lemma** *uIsInvar-pref-Conflict-isRevNth*: *uIsInvar* *pref-Conflict-isRevNth*

**proof** (*cases* *pref-Conflict-isRevNth* *rule*: *uIsInvar*)  
**case** (*uPref*  $s \text{ } \text{ confID } uID \text{ } p \text{ } \text{ paperID } \text{ pref}$ ) **thus** ?*case*  
**apply** (*auto simp*: *u-defs* *pref-Conflict-isRevNth-def*)  
**apply** (*frule* *isRevNth-geq-revPH*, *simp*)  
**apply** (*frule* *isRevNth-paperIDs*, *simp*)  
**apply** (*frule* *paperIDs-equals*, *simp*, *simp*, *force*)  
**done**

**qed** (*auto simp*: *u-defs* *pref-Conflict-isRevNth-def*)

**lemma** *uuIsInvar-pref-Conflict-isRevNth*: *uuIsInvar* *pref-Conflict-isRevNth*

**apply** (*cases* *pref-Conflict-isRevNth* *rule*: *uuIsInvar*)  
**by** (*auto simp*: *uu-defs* *pref-Conflict-isRevNth-def*)

**lemma** *invar-pref-Conflict-isRevNth*: *invar* *pref-Conflict-isRevNth*

**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*

**using** *cIsInvar-pref-Conflict-isRevNth* *uIsInvar-pref-Conflict-isRevNth* *uuIsInvar-pref-Conflict-isRevNth*  
**by** *auto*

**lemmas** *pref-Conflict-isRevNth1* =

*holdsIstate-invar* [*OF* *holdsIstate-pref-Conflict-isRevNth* *invar-pref-Conflict-isRevNth*]

**theorem** *pref-Conflict-isRevNth*:

**assumes**  $a$ : *reach*  $s$  **and**  $i$ : *pref*  $s \text{ } uID \text{ } papID = \text{ Conflict}$

**shows**  $\neg \text{isRevNth } s \text{ confID } uID \text{ papID } n$   
**using** *pref-Conflict-isRevNth1*[OF a] *i* **unfolding** *pref-Conflict-isRevNth-def* **by**  
*auto*

**corollary** *pref-Conflict-isRev*:  
**assumes** *a*: *reach s* **and** *i*: *pref s uID papID = Conflict*  
**shows**  $\neg \text{isRev } s \text{ confID } uID \text{ papID}$   
**using** *pref-Conflict-isRevNth*[OF a] *i* **unfolding** *isRev-def2* **by** *auto*

**corollary** *pref-isAut-isRevNth*:  
**assumes** *a*: *reach s* **and** *i*: *isAut s confID uID papID*  
**shows**  $\neg \text{isRevNth } s \text{ confID } uID \text{ papID } n$   
**using** *pref-Conflict-isRevNth*[OF a] *isAut-pref-Conflict*[OF a i] **by** *auto*

**corollary** *pref-isAut-isRev*:  
**assumes** *a*: *reach s* **and** *i*: *isAut s confID uID papID*  
**shows**  $\neg \text{isRev } s \text{ confID } uID \text{ papID}$   
**using** *pref-isAut-isRevNth*[OF a] *i* **unfolding** *isRev-def2* **by** *auto*

**definition** *isChair-isPC* :: *state*  $\Rightarrow$  *bool* **where**  
*isChair-isPC* *s*  $\equiv$   
 $\forall \text{ confID } uID. \text{isChair } s \text{ confID } uID \longrightarrow \text{isPC } s \text{ confID } uID$

**lemma** *holdsIstate-isChair-isPC*: *holdsIstate isChair-isPC*  
**unfolding** *IO-Automaton.holdsIstate-def* *istate-def* *istate-def* *isChair-isPC-def* **by**  
*auto*

**lemma** *cIsInvar-isChair-isPC*: *cIsInvar isChair-isPC*  
**apply** (*cases isChair-isPC* *rule: cIsInvar*)  
**by** (*auto simp: c-defs isChair-isPC-def*)

**lemma** *uIsInvar-isChair-isPC*: *uIsInvar isChair-isPC*  
**apply** (*cases isChair-isPC* *rule: uIsInvar*)  
**by** (*auto simp: u-defs isChair-isPC-def*)

**lemma** *uuIsInvar-isChair-isPC*: *uuIsInvar isChair-isPC*  
**apply** (*cases isChair-isPC* *rule: uuIsInvar*)  
**by** (*auto simp: uu-defs isChair-isPC-def*)

**lemma** *invar-isChair-isPC*: *invar isChair-isPC*  
**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*  
**using** *cIsInvar-isChair-isPC* *uIsInvar-isChair-isPC*  
*uuIsInvar-isChair-isPC* **by** *auto*

**lemmas** *isChair-isPC1* =  
*holdsIstate-invar*[OF *holdsIstate-isChair-isPC invar-isChair-isPC*]

**theorem** *isChair-isPC*:  
**assumes** *a*: reach *s* **and** *p*: isChair *s* *confID* *uID*  
**shows** *isPC* *s* *confID* *uID*  
**using** *isChair-isPC1*[*OF a*] *p* **unfolding** *isChair-isPC-def* **by** *auto*

**definition** *isRevNth-equals* :: state  $\Rightarrow$  bool **where**  
*isRevNth-equals* *s*  $\equiv$   
 $\forall$  *confID* *uID* *papID* *m* *n*.  
*isRevNth* *s* *confID* *uID* *papID* *m*  $\wedge$  *isRevNth* *s* *confID* *uID* *papID* *n*  
 $\longrightarrow$  *m* = *n*

**lemma** *holdsIstate-isRevNth-equals*: *holdsIstate* *isRevNth-equals*  
**unfolding** *IO-Automaton.holdsIstate-def* *istate-def* *istate-def* *isRevNth-equals-def*  
**by** *auto*

**lemma** *cIsInvar-isRevNth-equals*: *cIsInvar* *isRevNth-equals*  
**proof** (*cases isRevNth-equals* rule: *cIsInvar*)

**case** (*cReview* *s* *confID* *uID* *p* *papID* *uID'*)  
    **thus** ?*case* **by** (*fastforce* *simp* add: *c-defs isRevNth-equals-def isRev-def2*)  
**qed** (*auto* *simp*: *c-defs isRevNth-equals-def*)

**lemma** *uIsInvar-isRevNth-equals*: *uIsInvar* *isRevNth-equals*  
**apply** (*cases isRevNth-equals* rule: *uIsInvar*)  
**by** (*auto* *simp*: *u-defs isRevNth-equals-def*)

**lemma** *uuIsInvar-isRevNth-equals*: *uuIsInvar* *isRevNth-equals*  
**apply** (*cases isRevNth-equals* rule: *uuIsInvar*)  
**by** (*auto* *simp*: *uu-defs isRevNth-equals-def*)

**lemma** *invar-isRevNth-equals*: *invar* *isRevNth-equals*  
**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*  
**using** *cIsInvar-isRevNth-equals* *uIsInvar-isRevNth-equals*  
*uuIsInvar-isRevNth-equals* **by** *auto*

**lemmas** *isRevNth-equals1* =  
*holdsIstate-invar*[*OF holdsIstate-isRevNth-equals invar-isRevNth-equals*]

**theorem** *isRevNth-equals*:  
**assumes** *a*: reach *s* **and** *r*: *isRevNth* *s* *confID* *uID* *papID* *m* *isRevNth* *s* *confID* *uID*  
*papID* *n*  
**shows** *m* = *n*  
**using** *isRevNth-equals1*[*OF a*] *r* **unfolding** *isRevNth-equals-def* **by** *blast*

**corollary** *isRevNth-getReviewIndex*:  
**assumes** *a*: reach *s* **and** *r*: *isRevNth* *s* *confID* *uID* *papID* *n*  
**shows** *n* = *getReviewIndex* *s* *confID* *uID* *papID*  
**using** *isRevNth-equals*[*OF a*] *r*

**by** (*metis isRev-def2 isRev-def3*)

**definition** *isRevNth-less-length* :: *state*  $\Rightarrow$  *bool* **where**  
*isRevNth-less-length* *s*  $\equiv$   
 $\forall$  *confID uID papID n*.  
*isRevNth* *s confID uID papID n*  $\longrightarrow$  *n* < *length (reviewsPaper (paper s papID))*

**lemma** *holdsIstate-isRevNth-less-length*: *holdsIstate isRevNth-less-length*  
**unfolding** *IO-Automaton.holdsIstate-def istate-def istate-def isRevNth-less-length-def*  
**by** *auto*

**lemma** *cIsInvar-isRevNth-less-length*: *cIsInvar isRevNth-less-length*  
**apply**(*cases isRevNth-less-length rule: cIsInvar*)  
**by**(*fastforce simp: c-defs isRevNth-less-length-def*  
*isRevNth-getAllPaperIDs isRev-def2 isRevNth-paperIDs paperIDs-equals less-SucI*)+

**lemma** *uIsInvar-isRevNth-less-length*: *uIsInvar isRevNth-less-length*  
**apply**(*cases isRevNth-less-length rule: uIsInvar*)  
**by**(*fastforce simp: u-defs isRevNth-less-length-def*  
*isRevNth-getAllPaperIDs isRev-def2 isRevNth-paperIDs paperIDs-equals less-SucI*)+

**lemma** *uuIsInvar-isRevNth-less-length*: *uuIsInvar isRevNth-less-length*  
**apply** (*cases isRevNth-less-length rule: uuIsInvar*)  
**by**(*fastforce simp: uu-defs isRevNth-less-length-def*  
*isRevNth-getAllPaperIDs isRev-def2 isRevNth-paperIDs paperIDs-equals less-SucI*)+

**lemma** *invar-isRevNth-less-length*: *invar isRevNth-less-length*  
**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*  
**using** *cIsInvar-isRevNth-less-length uIsInvar-isRevNth-less-length*  
*uuIsInvar-isRevNth-less-length* **by** *auto*

**lemmas** *isRevNth-less-length1* =  
*holdsIstate-invar[OF holdsIstate-isRevNth-less-length invar-isRevNth-less-length]*

**theorem** *isRevNth-less-length*:  
**assumes** *reach s* **and** *isRevNth s cid uid pid n*  
**shows** *n* < *length (reviewsPaper (paper s pid))*  
**using** *isRevNth-less-length1* *assms* **unfolding** *isRevNth-less-length-def* **by** *blast*

**definition** *isRevNth-equalsU* :: *state*  $\Rightarrow$  *bool* **where**  
*isRevNth-equalsU* *s*  $\equiv$   
 $\forall$  *confID uID uID1 papID n*.  
*isRevNth* *s confID uID papID n*  $\wedge$  *isRevNth* *s confID uID1 papID n*  
 $\longrightarrow$  *uID = uID1*

**lemma** *holdsIstate-isRevNth-equalsU*: *holdsIstate isRevNth-equalsU*  
**unfolding** *IO-Automaton.holdsIstate-def istate-def istate-def isRevNth-equalsU-def*  
**by** *auto*

**lemma** *cIsInvar-isRevNth-equalsU*: *cIsInvar isRevNth-equalsU*

**apply** (*cases isRevNth-equalsU rule: cIsInvar*)  
**apply** (*fastforce simp: c-defs isRevNth-equalsU-def*)**+**

**proof** –

**fix** *s confID uID p papID uID'*

**assume** *s: reach s*

**and** *0: isRevNth-equalsU s e-createReview s confID uID p papID uID'*

**let** *?s' = createReview s confID uID p papID uID'*

**show** *isRevNth-equalsU ?s'*

**unfolding** *isRevNth-equalsU-def* **proof** *clarify*

**fix** *confIDa uIDa uID1 papIDa n*

**assume** *isRevNth ?s' confIDa uIDa papIDa n isRevNth ?s' confIDa uID1 papIDa*

*n*

**thus** *uIDa = uID1*

**apply** (*cases confIDa = confID  $\wedge$  papIDa = papID*)

**apply** (*cases uIDa = uID, cases uID1 = uID*)

**using** *s 0 isRevNth-less-length[OF s, of papID n]* **unfolding** *isRevNth-less-length-def*

**by** (*fastforce simp: c-defs isRevNth-equalsU-def*)**+**

**qed**

**qed**

**lemma** *uIsInvar-isRevNth-equalsU*: *uIsInvar isRevNth-equalsU*

**apply** (*cases isRevNth-equalsU rule: uIsInvar*)

**by** (*auto simp: u-defs isRevNth-equalsU-def*)

**lemma** *uuIsInvar-isRevNth-equalsU*: *uuIsInvar isRevNth-equalsU*

**apply** (*cases isRevNth-equalsU rule: uuIsInvar*)

**by** (*auto simp: uu-defs isRevNth-equalsU-def*)

**lemma** *invar-isRevNth-equalsU*: *invar isRevNth-equalsU*

**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*

**using** *cIsInvar-isRevNth-equalsU uIsInvar-isRevNth-equalsU*

*uuIsInvar-isRevNth-equalsU* **by** *auto*

**lemmas** *isRevNth-equalsU1 =*

*holdsIstate-invar[OF holdsIstate-isRevNth-equalsU invar-isRevNth-equalsU]*

**theorem** *isRevNth-equalsU*:

**assumes** *a: reach s* **and** *r: isRevNth s confID uID papID n isRevNth s confID*  
*uID1 papID n*

**shows** *uID = uID1*

**using** *isRevNth-equalsU1[OF a] r* **unfolding** *isRevNth-equalsU-def* **by** *blast*

**definition** *reviews-compact* :: *state  $\Rightarrow$  bool* **where**

*reviews-compact*  $s \equiv$   
 $\forall$  *confID* *paperID*  $n$ .  
 $paperID \in \in paperIDs\ s\ confID \wedge n < length\ (reviewsPaper\ (paper\ s\ paperID)) \longrightarrow$   
 $(\exists\ uid. isRevNth\ s\ confID\ uid\ paperID\ n)$

**lemma** *holdsIstate-reviews-compact*: *holdsIstate* *reviews-compact*  
**unfolding** *IO-Automaton.holdsIstate-def* *istate-def* *istate-def* *reviews-compact-def*  
**by** *auto*

**lemma** *cIsInvar-reviews-compact*: *cIsInvar* *reviews-compact*  
**apply**(*cases* *reviews-compact* *rule*: *cIsInvar*)  
**apply**(*auto* *simp*: *c-defs* *reviews-compact-def*  
*isRevNth-getAllPaperIDs* *isRev-def2* *isRevNth-paperIDs* *paperIDs-equals* *less-SucI*)  
**using** *paperIDs-confIDs*  
**apply** *fastforce*  
**apply** *metis*  
**apply** *metis*  
**apply** *metis*  
**using** *less-Suc-eq* **apply** *auto*[1]  
**apply** *metis*  
**done**

**lemma** *uIsInvar-reviews-compact*: *uIsInvar* *reviews-compact*  
**apply**(*cases* *reviews-compact* *rule*: *uIsInvar*)  
**by**(*fastforce* *simp*: *u-defs* *reviews-compact-def*  
*isRevNth-getAllPaperIDs* *isRev-def2* *isRevNth-paperIDs* *paperIDs-equals* *less-SucI*)+

**lemma** *uuIsInvar-reviews-compact*: *uuIsInvar* *reviews-compact*  
**apply** (*cases* *reviews-compact* *rule*: *uuIsInvar*)  
**by**(*fastforce* *simp*: *uu-defs* *reviews-compact-def*  
*isRevNth-getAllPaperIDs* *isRev-def2* *isRevNth-paperIDs* *paperIDs-equals* *less-SucI*)+

**lemma** *invar-reviews-compact*: *invar* *reviews-compact*  
**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*  
**using** *cIsInvar-reviews-compact* *uIsInvar-reviews-compact*  
*uuIsInvar-reviews-compact* **by** *auto*

**lemmas** *reviews-compact1* =  
*holdsIstate-invar*[*OF* *holdsIstate-reviews-compact* *invar-reviews-compact*]

**theorem** *reviews-compact*:  
**assumes** *reach*  $s$  **and**  $n < length\ (reviewsPaper\ (paper\ s\ pid))$   
**and**  $pid \in \in paperIDs\ s\ cid$   
**shows**  $\exists\ uid. isRevNth\ s\ cid\ uid\ pid\ n$   
**using** *reviews-compact1* *assms* **unfolding** *reviews-compact-def* **by** *blast*

**definition** *roles-nonrep* :: *state*  $\Rightarrow$  *bool* **where**

*roles-nonrep s*  $\equiv$   
 $\forall$  *confID uID*.  
*distinct (roles s confID uID)*

**lemma** *holdsIstate-roles-nonrep: holdsIstate roles-nonrep*  
**unfolding** *IO-Automaton.holdsIstate-def istate-def istate-def roles-nonrep-def* **by**  
*auto*

**lemma** *cIsInvar-roles-nonrep: cIsInvar roles-nonrep*  
**apply**(*cases roles-nonrep rule: cIsInvar*)  
**by** (*auto simp: c-defs roles-nonrep-def*  
*isRevNth-getAllPaperIDs isRev-def2 isRevNth-paperIDs paperIDs-equals less-SucI*)

**lemma** *uIsInvar-roles-nonrep: uIsInvar roles-nonrep*  
**apply**(*cases roles-nonrep rule: uIsInvar*)  
**by**(*fastforce simp: u-defs roles-nonrep-def*  
*isRevNth-getAllPaperIDs isRev-def2 isRevNth-paperIDs paperIDs-equals less-SucI*)+

**lemma** *uuIsInvar-roles-nonrep: uuIsInvar roles-nonrep*  
**apply** (*cases roles-nonrep rule: uuIsInvar*)  
**by**(*fastforce simp: uu-defs roles-nonrep-def*  
*isRevNth-getAllPaperIDs isRev-def2 isRevNth-paperIDs paperIDs-equals less-SucI*)+

**lemma** *invar-roles-nonrep: invar roles-nonrep*  
**unfolding** *invar-cIsInvar-uIsInvar-uuIsInvar*  
**using** *cIsInvar-roles-nonrep uIsInvar-roles-nonrep*  
*uuIsInvar-roles-nonrep* **by** *auto*

**lemmas** *roles-nonrep1 =*  
*holdsIstate-invar[OF holdsIstate-roles-nonrep invar-roles-nonrep]*

**theorem** *roles-nonrep:*  
**assumes** *reach s*  
**shows** *distinct (roles s confID uID)*  
**using** *roles-nonrep1 assms* **unfolding** *roles-nonrep-def* **by** *blast*

### 3.3 Properties of the step function

**lemma** *step-outErr-eq: step s a = (outErr, s')  $\implies$  s' = s*  
**apply** (*cases a*)  
  **subgoal for** *x1* **apply** (*cases x1, simp-all add: c-defs*) .  
  **subgoal for** *x2* **apply** (*cases x2, simp-all add: u-defs*) .  
  **subgoal for** *x3* **apply** (*cases x3, simp-all add: uu-defs*) .  
  **by** *auto*

**lemma** *phase-increases:*  
**assumes** *step s a = (ou, s')*  
**shows** *phase s cid  $\leq$  phase s' cid*  
**using** *assms*

**apply** (*cases a*)  
 subgoal for *x1* **apply**(*cases x1*) **apply**(*auto simp: c-defs*) .  
 subgoal for *x2* **apply**(*cases x2*) **apply**(*auto simp: u-defs*) .  
 subgoal for *x3* **apply**(*cases x3*) **apply**(*auto simp: uu-defs*) .  
 by *auto*

**lemma** *phase-increases2*: *phase s CID ≤ phase (snd (step s a)) CID*  
 by (*metis phase-increases snd-conv surj-pair*)

**lemma** *confIDs-mono*:  
 assumes *step s a = (ou,s')* and *cid ∈∈ confIDs s*  
 shows *cid ∈∈ confIDs s'*  
 using *assms*  
**apply** (*cases a*)  
 subgoal for *x1* **apply**(*cases x1*) **apply**(*auto simp: c-defs*) .  
 subgoal for *x2* **apply**(*cases x2*) **apply**(*auto simp: u-defs*) .  
 subgoal for *x3* **apply**(*cases x3*) **apply**(*auto simp: uu-defs*) .  
 by *auto*

**lemma** *userIDs-mono*:  
 assumes *step s a = (ou,s')* and *uid ∈∈ userIDs s*  
 shows *uid ∈∈ userIDs s'*  
 using *assms*  
**apply** (*cases a*)  
 subgoal for *x1* **apply**(*cases x1*) **apply**(*auto simp: c-defs*) .  
 subgoal for *x2* **apply**(*cases x2*) **apply**(*auto simp: u-defs*) .  
 subgoal for *x3* **apply**(*cases x3*) **apply**(*auto simp: uu-defs*) .  
 by *auto*

**lemma** *paperIDs-mono*:  
 assumes *step s a = (ou,s')* and *pid ∈∈ paperIDs s cid*  
 shows *pid ∈∈ paperIDs s' cid*  
 using *assms*  
**apply** (*cases a*)  
 subgoal for *x1* **apply**(*cases x1*) **apply**(*auto simp: c-defs*) .  
 subgoal for *x2* **apply**(*cases x2*) **apply**(*auto simp: u-defs*) .  
 subgoal for *x3* **apply**(*cases x3*) **apply**(*auto simp: uu-defs*) .  
 by *auto*

**lemma** *isPC-persistent*:  
 assumes *isPC s cid uid* and *step s a = (ou, s')*  
 shows *isPC s' cid uid*  
 using *assms* **apply** (*cases a*)  
 subgoal for *x1* **apply**(*cases x1*) **apply**(*auto simp: c-defs*) .  
 subgoal for *x2* **apply**(*cases x2*) **apply**(*auto simp: u-defs*) .  
 subgoal for *x3* **apply**(*cases x3*) **apply**(*auto simp: uu-defs*) .  
 by *auto*

**lemma** *isChair-persistent*:

**assumes**  $isChair\ s\ cid\ uid$  **and**  $step\ s\ a = (ou, s')$   
**shows**  $isChair\ s'\ cid\ uid$   
**using**  $assms$  **apply** ( $cases\ a$ )  
    **subgoal** **for**  $x1$  **apply**( $cases\ x1$ ) **apply**( $auto\ simp: c-defs$ ) .  
    **subgoal** **for**  $x2$  **apply**( $cases\ x2$ ) **apply**( $auto\ simp: u-defs$ ) .  
    **subgoal** **for**  $x3$  **apply**( $cases\ x3$ ) **apply**( $auto\ simp: uu-defs$ ) .  
**by**  $auto$

### 3.4 Action-safety properties

**lemma**  $pref\text{-}Conflict\text{-}disPH$ :

**assumes**  $reach\ s$  **and**  $pid \in \in paperIDs\ s\ cid$  **and**  $pref\ s\ uid\ pid \neq Conflict$  **and**  
 $phase\ s\ cid = disPH$

**and**  $step\ s\ a = (ou, s')$

**shows**  $pref\ s'\ uid\ pid \neq Conflict$

**proof**–

**have**  $1: cid \in \in confIDs\ s$  **using**  $assms$  **by** ( $metis\ geq\text{-}noPH\text{-}confIDs\ zero\text{-}less\text{-}Suc$ )

**thus**  $?thesis$  **using**  $assms$

**apply**( $cases\ a$ )

**subgoal** **for**  $x1$  **apply**( $cases\ x1, auto\ simp: c-defs\ getAllPaperIDs\text{-}def$ )

**apply** ( $metis\ Suc\text{-}inject\ Zero\text{-}not\text{-}Suc\ paperIDs\text{-}equals$ )

**apply** ( $metis\ Suc\text{-}inject\ Zero\text{-}not\text{-}Suc\ paperIDs\text{-}equals$ ) .

**subgoal** **for**  $x2$  **apply**( $cases\ x2, auto\ simp: u-defs$ )

**apply** ( $metis\ Suc\text{-}inject\ Zero\text{-}not\text{-}Suc\ paperIDs\text{-}equals$ ) .

**subgoal** **for**  $x3$  **apply**( $cases\ x3, auto\ simp: uu-defs$ ) .

**by**  $auto$

**qed**

**lemma**  $isRevNth\text{-}persistent$ :

**assumes**  $reach\ s$  **and**  $isRevNth\ s\ cid\ uid\ pid\ n$

**and**  $step\ s\ a = (ou, s')$

**shows**  $isRevNth\ s'\ cid\ uid\ pid\ n$

**using**  $assms$  **apply** ( $cases\ a$ )

**subgoal** **for**  $x1$  **apply**( $cases\ x1$ ) **apply**( $auto\ simp: c-defs\ roles\text{-}confIDs$ ) .

**subgoal** **for**  $x2$  **apply**( $cases\ x2$ ) **apply**( $auto\ simp: u-defs$ ) .

**subgoal** **for**  $x3$  **apply**( $cases\ x3$ ) **apply**( $auto\ simp: uu-defs$ ) .

**by**  $auto$

**lemma**  $nonempty\text{-}decsPaper\text{-}persist$ :

**assumes**  $s: reach\ s$

**and**  $pid: pid \in \in paperIDs\ s\ cid$

**and**  $decsPaper\ (paper\ s\ pid) \neq []$  **and**  $step\ s\ a = (ou, s')$

**shows**  $decsPaper\ (paper\ s'\ pid) \neq []$

**proof**–

**have**  $cid \in \in confIDs\ s$  **using**  $s\ pid$  **by** ( $metis\ paperIDs\text{-}confIDs$ )

**thus**  $?thesis$  **using**  $assms$  **apply**( $cases\ a$ )

**subgoal** **for**  $x1$  **apply**( $cases\ x1, auto\ simp: c-defs\ getAllPaperIDs\text{-}def$ ) .

**subgoal** **for**  $x2$  **apply**( $cases\ x2, auto\ simp: u-defs$ ) .

**subgoal** **for**  $x3$  **apply**( $cases\ x3, auto\ simp: uu-defs$ ) .

by auto  
qed

**lemma** *nonempty-reviews-persist*:

**assumes** *s*: reach *s*

**and** *r*: isRevNth *s* *cid* *uid* *pid* *n*

**and** (reviewsPaper (paper *s* *pid*))!*n* ≠ [] **and** step *s* *a* = (*ou*, *s'*)

**shows** (reviewsPaper (paper *s'* *pid*))!*n* ≠ []

**proof** –

have *pid*: *pid* ∈∈ paperIDs *s* *cid* **using** *s* *r* **by** (metis isRevNth-paperIDs)

have *cid*: *cid* ∈∈ confIDs *s* **using** *s* *pid* **by** (metis paperIDs-confIDs)

have *n*: *n* < length (reviewsPaper (paper *s* *pid*)) **using** *s* *r* **by** (metis isRevNth-less-length)

show ?thesis **using** *assms* *pid* *cid* *n* **apply**(cases *a*)

subgoal for *x1* **apply**(cases *x1*, auto simp: c-defs getAllPaperIDs-def) .

subgoal for *x2* **apply**(cases *x2*, auto simp: u-defs)

apply (metis not-Cons-self2 nth-list-update-eq nth-list-update-neq) .

subgoal for *x3* **apply**(cases *x3*, auto simp: uu-defs)

apply (metis list.distinct(1) nth-list-update-eq nth-list-update-neq) .

by auto

qed

**lemma** *revPH-pref-persists*:

**assumes** reach *s*

*pid* ∈∈ paperIDs *s* *cid* **and** phase *s* *cid* ≥ revPH

**and** step *s* *a* = (*ou*, *s'*)

**shows** pref *s'* *uid* *pid* = pref *s* *uid* *pid*

**using** *assms* **apply**(cases *a*)

subgoal for *x1* **apply**(cases *x1*) **apply**(auto simp: c-defs paperIDs-getAllPaperIDs)

using paperIDs-equals **apply** fastforce

using paperIDs-equals **apply** fastforce .

subgoal for *x2* **apply**(cases *x2*) **apply**(auto simp: u-defs)

using paperIDs-equals **apply** fastforce .

subgoal for *x3* **apply**(cases *x3*) **apply**(fastforce simp: uu-defs)+ .

by auto

### 3.5 Miscellaneous

**lemma** *updates-commute-paper*:

∧ uu. *s* (∧ confIDs := uu, paper := pp) = *s* (∧ paper := pp, confIDs := uu)

∧ uu. *s* (∧ conf := uu, paper := pp) = *s* (∧ paper := pp, conf := uu)

∧ uu. *s* (∧ userIDs := uu, paper := pp) = *s* (∧ paper := pp, userIDs := uu)

∧ uu. *s* (∧ pass := uu, paper := pp) = *s* (∧ paper := pp, pass := uu)

∧ uu. *s* (∧ user := uu, paper := pp) = *s* (∧ paper := pp, user := uu)

∧ uu. *s* (∧ roles := uu, paper := pp) = *s* (∧ paper := pp, roles := uu)

∧ uu. *s* (∧ paperIDs := uu, paper := pp) = *s* (∧ paper := pp, paperIDs := uu)

$\wedge uu. s \langle pref := uu, paper := pp \rangle = s \langle paper := pp, pref := uu \rangle$   
 $\wedge uu. s \langle voronkov := uu, paper := pp \rangle = s \langle paper := pp, voronkov := uu \rangle$   
 $\wedge uu. s \langle news := uu, paper := pp \rangle = s \langle paper := pp, news := uu \rangle$   
 $\wedge uu. s \langle phase := uu, paper := pp \rangle = s \langle paper := pp, phase := uu \rangle$   
**by** (auto intro: state.equality)

**lemma** *isAUT-imp-isAut*:  
**assumes** reach *s* **and**  $pid \in \in paperIDs\ s\ cid$  **and** *isAUT* *s uid pid*  
**shows** *isAut* *s cid uid pid*  
**by** (metis assms *isAUT-def isAut-paperIDs paperIDs-equals*)

**lemma** *isREVNth-imp-isRevNth*:  
**assumes** reach *s* **and**  $pid \in \in paperIDs\ s\ cid$  **and** *isREVNth* *s uid pid n*  
**shows** *isRevNth* *s cid uid pid n*  
**by** (metis assms *isREVNth-def isRevNth-paperIDs paperIDs-equals*)

**lemma** *phase-increases-validTrans*:  
**assumes** *validTrans* (*Trans* *s a ou s'*)  
**shows** *phase* *s cid*  $\leq$  *phase* *s' cid*  
**using** *assms* **apply**(*cases a*)  
  **subgoal** **for** *x1* **apply**(*cases x1, auto simp: c-defs split: if-splits*) .  
  **subgoal** **for** *x2* **apply**(*cases x2, auto simp: u-defs split: if-splits paper.splits*) .  
  **subgoal** **for** *x3* **apply**(*cases x3, auto simp: uu-defs split: if-splits paper.splits*) .  
  **by** *auto*

**lemma** *phase-increases-validTrans2*:  
**assumes** *validTrans* *tr*  
**shows** *phase* (*srcOf* *tr*) *cid*  $\leq$  *phase* (*tgtOf* *tr*) *cid*  
**using** *assms* *phase-increases-validTrans* **by** (*cases tr*) *auto*

**lemma** *phase-increases-trace*:  
**assumes** *vtr: valid* *tr* **and** *ij: i*  $\leq$  *j* **and** *j: j*  $<$  *length* *tr*  
**shows** *phase* (*srcOf* (*tr*!*i*)) *cid*  $\leq$  *phase* (*srcOf* (*tr*!*j*)) *cid*  
**proof**(*cases i < j*)  
**case** *False* **thus** *?thesis* **using** *ij* **by** *auto*  
**next**  
**case** *True* **thus** *?thesis*  
**using** *j* **proof**(*induction j*)  
  **case** (*Suc jj*)  
  **show** *?case*  
  **proof**(*cases jj = i*)  
    **case** *True*  
    **obtain** *tr1 tr2* **where** *tr: tr* = *tr1* @ (*tr*!*i*) # (*tr*!(*Suc jj*)) # *tr2*

**unfolding** *True* **by** (*metis Cons-nth-drop-Suc Suc.premis(2) Suc-lessD True id-take-nth-drop*)  
**hence** *validTrans (tr!i) ∧ tgtOf (tr!i) = srcOf (tr!(Suc jj))*  
**unfolding** *True* **by** (*metis Suc Suc-lessD True valid-validTrans-nth valid-validTrans-nth-srcOf-tgtOf vtr*)  
**thus** *?thesis* **using** *phase-increases-validTrans Suc* **by** (*cases tr!i*) *auto*  
**next**  
**case** *False* **hence** *1: i < jj ∧ jj < length tr* **using** *Suc* **by** *auto*  
**hence** *phase (srcOf (tr!i)) cid ≤ phase (srcOf (tr!jj)) cid* **using** *Suc* **by** *auto*  
**also have** *phase (srcOf (tr!jj)) cid ≤ phase (tgtOf (tr!jj)) cid*  
**using** *phase-increases-validTrans2* **by** (*metis 1 valid-validTrans-nth vtr*)  
**also have** *... = phase (srcOf (tr!(Suc jj))) cid*  
**by** (*metis Suc valid-validTrans-nth-srcOf-tgtOf vtr*)  
**finally show** *?thesis* .  
**qed**  
**qed** *auto*  
**qed**

**lemma** *phase-increases-trace-srcOf-tgtOf*:  
**assumes** *vtr: valid tr* **and** *ij: i ≤ j* **and** *j: j < length tr*  
**shows** *phase (srcOf (tr!i)) cid ≤ phase (tgtOf (tr!j)) cid*  
**using** *phase-increases-trace[OF assms]*  
**using** *j le-trans phase-increases-validTrans2 valid-validTrans-nth vtr* **by** *blast*

**lemma** *phase-increases-trace-srcOf-hd*:  
**assumes** *v: valid tr* **and** *l: length tr > 1* **and** *i < length tr*  
**shows** *phase (srcOf (hd tr)) cid ≤ phase (srcOf (tr!i)) cid*  
**using** *phase-increases-trace assms*  
**by** (*metis gr-implies-not0 hd-Cons-tl leI length-0-conv nth-Cons-0*)

**lemma** *phase-increases-trace-srcOf-last*:  
**assumes** *v: valid tr* **and** *l: length tr > 1* **and** *i: i < length tr*  
**shows** *phase (srcOf (tr!i)) cid ≤ phase (srcOf (last tr)) cid*  
**proof** –  
**have** *1: last tr = tr!(length tr – 1)*  
**by** (*metis i last-conv-nth list.size(3) not-less0*)  
**show** *?thesis* **unfolding** *1* **using** *assms*  
**by** (*metis Suc-diff-1 Suc-leI Suc-le-mono gr-implies-not0 length-0-conv length-greater-0-conv lessI phase-increases-trace*)  
**qed**

**lemma** *phase-increases-trace-srcOf-tgtOf-last*:  
**assumes** *v: valid tr* **and** *l: length tr > 1* **and** *i: i < length tr*  
**shows** *phase (srcOf (tr!i)) cid ≤ phase (tgtOf (last tr)) cid*  
**proof** –  
**have** *1: last tr = tr!(length tr – 1)*  
**by** (*metis i last-conv-nth list.size(3) not-less0*)  
**have** *phase (srcOf (tr!i)) cid ≤ phase (srcOf (last tr)) cid* **using**  
*phase-increases-trace-srcOf-last[OF assms]* .

**also have**  $\dots \leq \text{phase } (\text{tgtOf } (\text{last } \text{tr})) \text{ cid}$  **unfolding 1**  
**by** (*metis Suc-le-D diff-Suc-1 l lessI less-eq-Suc-le phase-increases-validTrans2 v valid-validTrans-nth*)  
**finally show** *?thesis* **by** (*simp add: le-funD*)  
**qed**

**lemma** *valid-tgtPf-last-srcOf*:  
**assumes** *valid tr* **and**  $s \in \text{map } \text{tgtOf } \text{tr}$   
**shows**  $s = \text{tgtOf } (\text{last } \text{tr}) \vee s \in \text{map } \text{srcOf } \text{tr}$   
**using** *assms* **by** *induction auto*

**lemma** *phase-constant*:  
**assumes**  $v$ : *valid tr* **and**  $l$ :  $\text{length } \text{tr} > 0$  **and**  
 $ph$ :  $\text{phase } (\text{srcOf } (\text{hd } \text{tr})) \text{ cid} = \text{phase } (\text{tgtOf } (\text{last } \text{tr})) \text{ cid}$   
**shows**  $\text{set } (\text{map } (\lambda \text{trn. } \text{phase } (\text{srcOf } \text{trn}) \text{ cid}) \text{tr}) \subseteq \{\text{phase } (\text{srcOf } (\text{hd } \text{tr})) \text{ cid}\}$   
 $\wedge$   
 $\text{set } (\text{map } (\lambda \text{trn. } \text{phase } (\text{tgtOf } \text{trn}) \text{ cid}) \text{tr}) \subseteq \{\text{phase } (\text{srcOf } (\text{hd } \text{tr})) \text{ cid}\}$   
**proof**(*cases length tr > 1*)  
**case** *False*  
**then obtain**  $\text{trn}$  **where**  $\text{tr} = [\text{trn}]$  **using**  $l$  **by** (*cases tr*) *auto*  
**show** *?thesis* **using** *assms* **unfolding**  $\text{tr}$  **by** *auto*  
**next**  
**case** *True* **note**  $l = \text{True}$   
**show** *?thesis* **proof** *safe*  
 $\{\text{fix } ph \text{ assume } ph \in \text{map } (\lambda \text{trn. } \text{phase } (\text{srcOf } \text{trn}) \text{ cid}) \text{tr}$   
 $\text{then obtain } i \text{ where } i: i < \text{length } \text{tr} \text{ and } phe: ph = \text{phase } (\text{srcOf } (\text{tr}!i)) \text{ cid}$   
 $\text{by } (\text{smt } (\text{verit}) \text{ comp-apply in-set-conv-nth length-map nth-map})$   
 $\text{have } \text{phase } (\text{srcOf } (\text{hd } \text{tr})) \text{ cid} \leq ph$   
 $\text{unfolding } phe \text{ using } v \text{ l } i \text{ phase-increases-trace-srcOf-hd}$  **by** *blast*  
 $\text{moreover have } ph \leq \text{phase } (\text{tgtOf } (\text{last } \text{tr})) \text{ cid}$   
 $\text{unfolding } phe \text{ using } v \text{ l } i \text{ phase-increases-trace-srcOf-tgtOf-last}$  **by** *auto*  
 $\text{ultimately show } ph = \text{phase } (\text{srcOf } (\text{hd } \text{tr})) \text{ cid}$  **using**  $ph$  **by** *simp*  
 $\}$  **note**  $0 = \text{this}$   
 $\text{fix } ph \text{ assume } ph \in \text{map } (\lambda \text{trn. } \text{phase } (\text{tgtOf } \text{trn}) \text{ cid}) \text{tr}$   
 $\text{then obtain } s \text{ where } s \in \text{map } \text{tgtOf } \text{tr} \text{ and } phs: ph = \text{phase } s \text{ cid}$  **by** *auto*  
 $\text{hence } s = \text{tgtOf } (\text{last } \text{tr}) \vee s \in \text{map } \text{srcOf } \text{tr}$  **using** *valid-tgtPf-last-srcOf*[*OF*  
 $v$ ] **by** *auto*  
 $\text{thus } ph = \text{phase } (\text{srcOf } (\text{hd } \text{tr})) \text{ cid}$  **using**  $0[\text{of } ph]$   $ph$  **unfolding**  $phs$  **by** *auto*  
**qed**  
**qed**

**lemma** *phase-cases*:  
**assumes**  $\text{step } s \text{ a} = (\text{ou}, s')$   
**obtains**  $(\text{noPH}) \neg \text{cid} \in \text{confIDs } s \vee \text{phase } s \text{ cid} = \text{noPH}$

$| (\text{Id}) \text{phase } s' \text{ cid} = \text{phase } s \text{ cid}$   
 $| (\text{Upd}) \text{uid } p \text{ ph}$  **where**  $\text{phase } s' \text{ cid} = ph \text{ a} = \text{Uact } (\text{uPhase } \text{cid } \text{uid } p \text{ ph})$   
 $e\text{-updatePhase } s \text{ cid } \text{uid } p \text{ ph}$   
**using** *assms* **proof** (*cases a*)

```

case (Cact ca)
then show thesis using assms
  by (cases ca) (auto simp: c-defs split: if-splits intro: that)
next
case (Uact ua)
then show thesis using assms
  apply (cases ua)
  subgoal by (auto simp: u-defs split: if-splits paper.splits intro: that)
  subgoal for x21 apply(cases x21 = cid)
    by (auto simp: u-defs split: if-splits paper.splits intro: that)
  subgoal for x31 apply(cases cid = x31)
    by (auto simp: u-defs split: if-splits paper.splits intro: that)
  by (auto simp: u-defs split: if-splits paper.splits intro: that)
next
case (UUact uua)
  then show thesis using assms by (cases uua) (auto simp: uu-defs split: if-splits
paper.splits intro: that)
qed auto

```

```

lemma phase-mono: reachFrom s s'  $\implies$  phase s cid  $\leq$  phase s' cid
proof (induction rule: reachFrom-step-induct)
  case (Step s' a ou s'')
    then show ?case
      proof (cases a)
        case (Cact cAct) with Step show ?thesis by (cases cAct) (auto simp add:
c-defs split: if-splits) next
        case (Uact uAct) with Step show ?thesis by (cases uAct) (auto simp add:
u-defs split: if-splits paper.split) next
        case (UUact uAct) with Step show ?thesis by (cases uAct) (auto simp add:
uu-defs split: if-splits paper.split)
      qed (auto)
    qed (auto)

```

```

lemma validTrans-rAct-lAct-srcOf-tgtOf:
assumes validTrans trn
and actOf trn = Ract rAct  $\vee$  actOf trn = Lact lAct
shows tgtOf trn = srcOf trn
using assms by (cases trn) auto

```

```

lemma valid-rAct-lAct-srcOf-tgtOf:
assumes valid tr
and  $\bigwedge a. a \in \in \text{map actOf tr} \implies (\exists rAct. a = Ract rAct) \vee (\exists lAct. a = Lact$ 
lAct)
shows srcOf ' (set tr)  $\subseteq$  {srcOf (hd tr)}
using assms by (induction) (simp-all, metis validTrans-rAct-lAct-srcOf-tgtOf)

```

```

lemma validFrom-rAct-lAct-srcOf-tgtOf:
assumes validFrom s tr
and  $\bigwedge a. a \in \in \text{map actOf tr} \implies (\exists rAct. a = Ract rAct) \vee (\exists lAct. a = Lact$ 

```

*lAct*)  
**shows** *srcOf* ' (*set tr*)  $\subseteq$  {*s*}  
**using** *assms valid-rAct-lAct-srcOf-tgtOf unfolding validFrom-def* **by** *auto*

**lemma** *tgtOf-last-traceOf-Ract-Lact[simp]*:  
**assumes** *al*  $\neq$  [] *set al*  $\subseteq$  *range Ract*  $\cup$  *range Lact*  
**shows** *tgtOf* (*last* (*traceOf s al*)) = *s*  
**using** *assms* **by** (*induction al arbitrary: s*) *auto*

**lemma** *paperIDs-cases*:  
**assumes** *step s a* = (*ou, s'*)  
**obtains** (*Id*) *paperIDs s' cid* = *paperIDs s cid*  
| (*Create*) *cid uid p pid tit ab* **where**  
*paperIDs s' cid* = *pid* # *paperIDs s cid* *a* = *Cact* (*cPaper cid uid p pid*  
*tit ab*)  
*e-createPaper s cid uid p pid tit ab*  
**using** *assms* **proof** (*cases a*)  
**case** (*Cact ca*)  
**then show** *thesis* **using** *assms*  
**by** (*cases ca*) (*auto simp: c-defs split: if-splits intro: that*)  
**next**  
**case** (*Uact ua*)  
**then show** *thesis* **using** *assms*  
**by** (*cases ua*) (*auto simp: u-defs split: if-splits paper.splits intro: that*)  
**next**  
**case** (*UUact ua*)  
**then show** *thesis* **using** *assms*  
**by** (*cases ua*) (*auto simp: uu-defs split: if-splits paper.splits intro: that*)  
**qed** *auto*

**lemma** *paperIDs-decPH-const*:  
**assumes** *s: step s a* = (*ou, s'*) **and** *phase s cid* > *subPH*  
**shows** *paperIDs s' cid* = *paperIDs s cid*  
**using** *assms*  
**apply** (*elim paperIDs-cases[where cid = cid]*)  
**subgoal** .  
**subgoal for** *cida*  
**apply**(*cases cida = cid, auto*)  
**using** *s* **by** (*auto simp: c-defs*) .

**end**  
**theory** *Observation-Setup*  
**imports** *Safety-Properties*  
**begin**

## 4 Observation setup for confidentiality properties

The observation infrastructure, consisting of a discriminator  $\gamma$  and a selector  $g$ , is the same for all our confidentiality properties. Namely, we fix a group UIDs of users, and consider the actions and outputs of these users.

```
consts UIDs :: userID set

type-synonym obs = act * out

fun  $\gamma$  :: (state,act,out) trans  $\Rightarrow$  bool where
 $\gamma$  (Trans - a - -) = (userOfA a  $\in$  UIDs)

fun  $g$  :: (state,act,out) trans  $\Rightarrow$  obs where
 $g$  (Trans - a ou -) = (a,ou)

end
theory Paper-Intro
imports ../Safety-Properties
begin
```

## 5 Paper Confidentiality

In this section, we prove confidentiality properties for the papers submitted to a conference. The secrets (values) of interest are therefore the different versions of a given paper (with identifier PID) uploaded into the system.

The two properties that we prove represent points of “compromise” between the strength of the declassification bound and that of the declassification trigger. Let

- T1 denote “the paper’s authorship”
- T2 denote “PC membership and the conference having reached the bidding phase”

The two bound-trigger combinations are:

- weak trigger (T1 or T2) paired with strong bound (nothing can be learned, save for some harmless information, namely the non-existence of any upload);
- strong trigger (T1) paired with weak bound (allowing to learn the last submitted version of the paper (but nothing more than that)).

**end**

```

theory Paper-Value-Setup
imports Paper-Intro
begin

```

```

consts PID :: paperID

```

## 5.1 Preliminaries

```

declare updates-commute-paper[simp]

```

```

fun eqButC :: paper  $\Rightarrow$  paper  $\Rightarrow$  bool where
eqButC (Paper name info ct reviews dis decs )
  (Paper name1 info1 ct1 reviews1 dis1 decs1) =
  (name = name1  $\wedge$  info = info1  $\wedge$  reviews = reviews1  $\wedge$  dis = dis1  $\wedge$  decs =
  decs1)

```

```

lemma eqButC:
eqButC pap pap1 =
  (titlePaper pap = titlePaper pap1  $\wedge$  abstractPaper pap = abstractPaper pap1  $\wedge$ 
  reviewsPaper pap = reviewsPaper pap1  $\wedge$  disPaper pap = disPaper pap1  $\wedge$  decsPaper pap = decsPaper pap1)
by(cases pap, cases pap1, auto)

```

```

lemma eqButC-eq[simp,intro!]: eqButC pap pap
by(cases pap) auto

```

```

lemma eqButC-sym:
assumes eqButC pap pap1
shows eqButC pap1 pap
apply(cases pap, cases pap1)
using assms by auto

```

```

lemma eqButC-trans:
assumes eqButC pap pap1 and eqButC pap1 pap2
shows eqButC pap pap2
apply(cases pap, cases pap1, cases pap2)
using assms by auto

```

```

definition eeqButPID where
eeqButPID paps paps1  $\equiv$ 
 $\forall$  pid. if pid = PID then eqButC (paps pid) (paps1 pid) else paps pid = paps1 pid

```

```

lemma eeqButPID-eeq[simp,intro!]: eeqButPID s s
unfolding eeqButPID-def by auto

```

```

lemma eeqButPID-sym:

```

**assumes** *eqButPID s s1* **shows** *eqButPID s1 s*  
**using** *assms eqButC-sym unfolding eqButPID-def* **by** *auto*

**lemma** *eqButPID-trans*:  
**assumes** *eqButPID s s1* **and** *eqButPID s1 s2* **shows** *eqButPID s s2*  
**using** *assms eqButC-trans unfolding eqButPID-def* **by** *simp blast*

**lemma** *eqButPID-imp*:  
*eqButPID paps paps1*  $\implies$  *eqButC (paps PID) (paps1 PID)*  
 $\llbracket \text{eqButPID paps paps1}; \text{pid} \neq \text{PID} \rrbracket \implies \text{paps pid} = \text{paps1 pid}$   
**unfolding** *eqButPID-def* **by** *auto*

**lemma** *eqButPID-cong*:  
**assumes** *eqButPID paps paps1*  
**and** *pid = PID*  $\implies$  *eqButC uu uu1*  
**and** *pid  $\neq$  PID*  $\implies$  *uu = uu1*  
**shows** *eqButPID (paps (pid := uu)) (paps1 (pid := uu1))*  
**using** *assms unfolding eqButPID-def* **by** *auto*

**lemma** *eqButPID-RDD*:  
*eqButPID paps paps1*  $\implies$   
*titlePaper (paps PID) = titlePaper (paps1 PID)*  $\wedge$   
*abstractPaper (paps PID) = abstractPaper (paps1 PID)*  $\wedge$   
*reviewsPaper (paps PID) = reviewsPaper (paps1 PID)*  $\wedge$   
*disPaper (paps PID) = disPaper (paps1 PID)*  $\wedge$   
*decsPaper (paps PID) = decsPaper (paps1 PID)*  
**using** *eqButPID-def unfolding eqButC* **by** *auto*

**definition** *eqButPID* :: *state*  $\Rightarrow$  *state*  $\Rightarrow$  *bool* **where**  
*eqButPID s s1*  $\equiv$   
*confIDs s = confIDs s1*  $\wedge$  *conf s = conf s1*  $\wedge$   
*userIDs s = userIDs s1*  $\wedge$  *pass s = pass s1*  $\wedge$  *user s = user s1*  $\wedge$  *roles s = roles*  
*s1*  $\wedge$   
*paperIDs s = paperIDs s1*  
 $\wedge$   
*eqButPID (paper s) (paper s1)*  
 $\wedge$   
*pref s = pref s1*  $\wedge$   
*voronkov s = voronkov s1*  $\wedge$   
*news s = news s1*  $\wedge$  *phase s = phase s1*

**lemma** *eqButPID-eq[simp,intro!]*: *eqButPID s s*  
**unfolding** *eqButPID-def* **by** *auto*

**lemma** *eqButPID-sym*:  
**assumes** *eqButPID s s1* **shows** *eqButPID s1 s*  
**using** *assms eqButPID-sym unfolding eqButPID-def* **by** *auto*

**lemma** *eqButPID-trans*:  
**assumes** *eqButPID s s1* **and** *eqButPID s1 s2* **shows** *eqButPID s s2*  
**using** *assms eqButPID-trans* **unfolding** *eqButPID-def* **by** *auto*

**lemma** *eqButPID-imp*:  
*eqButPID s s1*  $\implies$   
 $confIDs\ s = confIDs\ s1 \wedge conf\ s = conf\ s1 \wedge$   
 $userIDs\ s = userIDs\ s1 \wedge pass\ s = pass\ s1 \wedge user\ s = user\ s1 \wedge roles\ s = roles$   
 $s1 \wedge$   
 $paperIDs\ s = paperIDs\ s1$   
 $\wedge$   
 $eeqButPID\ (paper\ s)\ (paper\ s1)$   
 $\wedge$   
 $pref\ s = pref\ s1 \wedge$   
 $voronkov\ s = voronkov\ s1 \wedge$   
 $news\ s = news\ s1 \wedge phase\ s = phase\ s1 \wedge$   
 $getAllPaperIDs\ s = getAllPaperIDs\ s1 \wedge$   
 $isRev\ s\ cid\ uid\ pid = isRev\ s1\ cid\ uid\ pid \wedge$   
 $getReviewIndex\ s\ cid\ uid\ pid = getReviewIndex\ s1\ cid\ uid\ pid \wedge$   
 $getRevRole\ s\ cid\ uid\ pid = getRevRole\ s1\ cid\ uid\ pid$   
**unfolding** *eqButPID-def* *getAllPaperIDs-def*  
**unfolding** *isRev-def* *getReviewIndex-def* *getRevRole-def* **by** *auto*

**lemma** *eqButPID-imp1*:  
*eqButPID s s1*  $\implies$  *eqButC (paper s pid) (paper s1 pid)*  
*eqButPID s s1*  $\implies$   $pid \neq PID \vee PID \neq pid \implies$   
 $paper\ s\ pid = paper\ s1\ pid \wedge$   
 $getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$   
**unfolding** *eqButPID-def* *getNthReview-def* *eqButPID-def*  
**apply** *auto*  
**by** (*metis eqButC-eq*)

**lemma** *eqButPID-imp2*:  
**assumes** *eqButPID s s1* **and**  $pid \neq PID \vee PID \neq pid$   
**shows**  $getReviewersReviews\ s\ cid\ pid = getReviewersReviews\ s1\ cid\ pid$   
**proof** –  
**have**  
 $(\lambda uID. \text{if } isRev\ s\ cid\ uID\ pid \text{ then } [(uID, getNthReview\ s\ pid\ (getReviewIndex\ s\ cid\ uID\ pid))] \text{ else } []]) =$   
 $(\lambda uID. \text{if } isRev\ s1\ cid\ uID\ pid \text{ then } [(uID, getNthReview\ s1\ pid\ (getReviewIndex\ s1\ cid\ uID\ pid))] \text{ else } []])$   
**apply**(*rule ext*)  
**using** *assms* **by** (*auto simp: eqButPID-imp eqButPID-imp1*)  
**thus** *?thesis* **unfolding** *getReviewersReviews-def* **using** *assms* **by** (*simp add: eqButPID-imp*)  
**qed**

**lemma** *eqButPID-RDD*:

*eqButPID s s1*  $\implies$   
*titlePaper* (*paper s PID*) = *titlePaper* (*paper s1 PID*)  $\wedge$   
*abstractPaper* (*paper s PID*) = *abstractPaper* (*paper s1 PID*)  $\wedge$   
*reviewsPaper* (*paper s PID*) = *reviewsPaper* (*paper s1 PID*)  $\wedge$   
*disPaper* (*paper s PID*) = *disPaper* (*paper s1 PID*)  $\wedge$   
*decsPaper* (*paper s PID*) = *decsPaper* (*paper s1 PID*)

**using** *eqButPID-imp eeqButPID-RDD* **by** *auto*

**lemma** *eqButPID-cong[simp, intro]*:

$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|confIDs := uu1\!|)) (s1 (\!|confIDs := uu2\!|))$   
 $\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|conf := uu1\!|)) (s1 (\!|conf := uu2\!|))$

$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|userIDs := uu1\!|)) (s1 (\!|userIDs := uu2\!|))$   
 $\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|pass := uu1\!|)) (s1 (\!|pass := uu2\!|))$   
 $\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|user := uu1\!|)) (s1 (\!|user := uu2\!|))$   
 $\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|roles := uu1\!|)) (s1 (\!|roles := uu2\!|))$

$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|paperIDs := uu1\!|)) (s1 (\!|paperIDs := uu2\!|))$   
 $\bigwedge uu1 uu2. eqButPID s s1 \implies eeqButPID uu1 uu2 \implies eqButPID (s (\!|paper := uu1\!|)) (s1 (\!|paper := uu2\!|))$

$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|pref := uu1\!|)) (s1 (\!|pref := uu2\!|))$   
 $\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|voronkov := uu1\!|)) (s1 (\!|voronkov := uu2\!|))$   
 $\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|news := uu1\!|)) (s1 (\!|news := uu2\!|))$   
 $\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|phase := uu1\!|)) (s1 (\!|phase := uu2\!|))$

**unfolding** *eqButPID-def* **by** *auto*

**lemma** *eqButPID-Paper*:

**assumes** *s's1'*: *eqButPID s s1*

**and** *paper s pid* = *Paper title abstract pc reviews dis decs*

**and** *paper s1 pid* = *Paper title1 abstract1 pc1 reviews1 dis1 decs1*

**shows** *title* = *title1*  $\wedge$  *abstract* = *abstract1*  $\wedge$  *reviews* = *reviews1*  $\wedge$  *dis* = *dis1*  $\wedge$  *decs* = *decs1*

**using** *assms* **unfolding** *eqButPID-def* **apply** (*auto simp: eqButC eeqButPID-def*)  
**by** (*metis titlePaper.simps abstractPaper.simps reviewsPaper.simps disPaper.simps decsPaper.simps*) $+$

**definition**  $NOSIMP\ a \equiv a$   
**lemma** [cong]:  $NOSIMP\ a = NOSIMP\ a$  **by** *simp*

**lemma** *eqButPID-paper*:  
**assumes** *eqButPID s s1*  
**shows**  $paper\ s = (paper\ s1)(PID :=$   
 $\quad Paper\ (titlePaper\ (paper\ s1\ PID))$   
 $\quad (abstractPaper\ (paper\ s1\ PID))$   
 $\quad (contentPaper\ (NOSIMP\ (paper\ s\ PID)))$   
 $\quad (reviewsPaper\ (paper\ s1\ PID))$   
 $\quad (disPaper\ (paper\ s1\ PID))$   
 $\quad (decsPaper\ (paper\ s1\ PID))$   
 $\quad )$   
**apply** (*rule sym*)  
**using** *assms unfolding NOSIMP-def eqButPID-def eqqButPID-def*  
**apply** (*intro ext*)  
**apply** *simp*  
**apply** (*cases paper s1 PID, simp-all*)  
**apply** (*cases paper s PID, simp-all*)  
**done**

**lemmas** *eqButPID-simps = eqButPID-imp eqButPID-paper*

## 5.2 Value Setup

**type-synonym** *value = pcontent*

**fun**  $\varphi :: (state,act,out)\ trans \Rightarrow bool$  **where**  
 $\varphi\ (Trans\ -\ (Uact\ (uPaperC\ cid\ uid\ p\ pid\ ct))\ ou\ -) = (pid = PID \wedge ou = outOK)$   
 $\quad |$   
 $\varphi\ - = False$

**lemma**  $\varphi$ -def2:  
 $\varphi\ (Trans\ s\ a\ ou\ s') = (\exists\ cid\ uid\ p\ ct.\ a = Uact\ (uPaperC\ cid\ uid\ p\ PID\ ct) \wedge ou = outOK)$   
**proof** (*cases a*)  
 $\quad$  **case** (*Uact x2*)  
 $\quad$  **then show** *?thesis* **by** (*cases x2; auto*)  
**qed** *auto*

**fun**  $f :: (state,act,out)\ trans \Rightarrow value$  **where**  
 $f\ (Trans\ -\ (Uact\ (uPaperC\ cid\ uid\ p\ pid\ ct))\ -) = ct$

**lemma** *Uact-uPaperC-step-eqButPID*:  
**assumes**  $a = Uact\ (uPaperC\ cid\ uid\ p\ PID\ ct)$   
**and**  $step\ s\ a = (ou, s')$   
**shows** *eqButPID s s'*

**using** *assms* **unfolding** *eqButPID-def* *eeqButPID-def* **by** (*auto simp: u-defs*)

**lemma** *φ-step-eqButPID*:

**assumes**  $\varphi: \varphi$  (*Trans s a ou s'*)

**and** *step*: *step s a = (ou, s')*

**shows** *eqButPID s s'*

**using**  $\varphi$  *Uact-uPaperC-step-eqButPID[OF - s]* **unfolding**  $\varphi$ -*def2* **by** *blast*

**lemma** *eqButPID-step*:

**assumes** *s's1'*: *eqButPID s s1*

**and** *step*: *step s a = (ou, s')*

**and** *step1*: *step s1 a = (ou1, s1')*

**shows** *eqButPID s' s1'*

**proof** –

**note** *eqs* = *eqButPID-imp[OF s's1']*

**note** *eqs'* = *eqButPID-imp1[OF s's1']*

**note** *simps[simp]* = *c-defs u-defs uu-defs r-defs l-defs Paper-dest-conv eqButPID-def eeqButPID-def eqButC*

**note** \* = *step step1 eqs eqs'*

**then show** *?thesis*

**proof** (*cases a*)

**case** (*Cact x1*)

**then show** *?thesis* **using** \* **by** (*cases x1; auto*)

**next**

**case** (*Uact x2*)

**then show** *?thesis* **using** \* **by** (*cases x2; auto*)

**next**

**case** (*UUact x3*)

**then show** *?thesis* **using** \* **by** (*cases x3; auto*)

**qed** *auto*

**qed**

**lemma** *eqButPID-step-φ-imp*:

**assumes** *s's1'*: *eqButPID s s1*

**and** *step*: *step s a = (ou, s')* **and** *step1*: *step s1 a = (ou1, s1')*

**and**  $\varphi: \varphi$  (*Trans s a ou s'*)

**shows**  $\varphi$  (*Trans s1 a ou1 s1'*)

**using** *assms* **unfolding**  $\varphi$ -*def2* **by** (*auto simp add: u-defs eqButPID-imp*)

**lemma** *eqButPID-step-φ*:

**assumes** *s's1'*: *eqButPID s s1*

**and** *step*: *step s a = (ou, s')* **and** *step1*: *step s1 a = (ou1, s1')*

**shows**  $\varphi$  (*Trans s a ou s'*) =  $\varphi$  (*Trans s1 a ou1 s1'*)

**by** (*metis eqButPID-step-φ-imp eqButPID-sym assms*)

**end**

```

theory Paper-Aut-PC
imports ../Observation-Setup Paper-Value-Setup Bounded-Deducibility-Security.Compositional-Reasoning
begin

```

### 5.3 Confidentiality protection from users who are not the paper's authors or PC members

We verify the following property:

A group of users UIDs learns nothing about the various uploads of a paper PID (save for the non-existence of any upload) unless/until one of the following occurs:

- a user in UIDs becomes the paper's author or
- a user in UIDs becomes a PC member in the paper's conference and the conference moves to the bidding phase.

```

fun T :: (state,act,out) trans  $\Rightarrow$  bool where
  T (Trans - - ou s') =
    ( $\exists$  uid  $\in$  UIDs.
      isAUT s' uid PID  $\vee$ 
      ( $\exists$  cid. PID  $\in$  paperIDs s' cid  $\wedge$  isPC s' cid uid  $\wedge$  phase s' cid  $\geq$  bidPH)
    )

```

```

declare T.simps [simp del]

```

```

definition B :: value list  $\Rightarrow$  value list  $\Rightarrow$  bool where
  B vl vl1  $\equiv$  vl  $\neq$  []

```

```

interpretation BD-Security-IO where
  istate = istate and step = step and
   $\varphi = \varphi$  and  $f = f$  and  $\gamma = \gamma$  and  $g = g$  and  $T = T$  and  $B = B$ 
done

```

```

lemma reachNT-non-isAut-isPC-isChair:

```

```

assumes reachNT s and uid  $\in$  UIDs

```

```

shows

```

```

 $\neg$  isAut s cid uid PID  $\wedge$ 
  (isPC s cid uid  $\longrightarrow$   $\neg$  PID  $\in$  paperIDs s cid  $\vee$  phase s cid  $\leq$  subPH)  $\wedge$ 
  (isChair s cid uid  $\longrightarrow$   $\neg$  PID  $\in$  paperIDs s cid  $\vee$  phase s cid  $\leq$  subPH)

```

```

using assms

```

```

apply (cases rule: reachNT-state-cases)

```

```

apply (auto simp: istate-def)[]

```

```

apply clarsimp

```

```

by (simp add: T.simps assms(1) isAUT-def isChair-isPC not-less-eq-eq reachNT-reach)

```

```

lemma P-φ-γ:
assumes 1: reachNT s and 2: step s a = (ou,s') φ (Trans s a ou s')
shows  $\neg \gamma$  (Trans s a ou s')
using reachNT-non-isAut-isPC-isChair[OF 1] 2 unfolding T.simps φ-def2
by (auto simp add: u-defs)

major

lemma eqButPID-step-out:
assumes s's1': eqButPID s s1
and step: step s a = (ou,s') and step1: step s1 a = (ou1,s1')
and sT: reachNT s and s1: reach s1
and PID: PID ∈∈ paperIDs s cid
and UIDs: userOfA a ∈ UIDs
shows ou = ou1
proof –
note Inv = reachNT-non-isAut-isPC-isChair[OF sT UIDs]
note eqs = eqButPID-imp[OF s's1']
note eqs' = eqButPID-imp1[OF s's1']
note s = reachNT-reach[OF sT]

note simps[simp] = c-defs u-defs uu-defs r-defs l-defs Paper-dest-conv eqBut-
PID-def eqButPID-def eqButC
note * = step step1 eqs eqs' s s1 PID UIDs paperIDs-equals[OF s] Inv

show ?thesis
proof (cases a)
case (Cact x1)
then show ?thesis using * by (cases x1; auto)
next
case (Uact x2)
then show ?thesis using * by (cases x2; auto)
next
case (UUact x3)
then show ?thesis using * by (cases x3; auto)
next
case (Ract x4)
with * show ?thesis
proof (cases x4)
case (rPaperC x61 x62 x63 x64)
then show ?thesis using * Ract by (clarsimp; metis not-less-eq-eq)
next
case (rMyReview x81 x82 x83 x84)
then show ?thesis using * Ract by (auto simp: getNthReview-def)
next
case (rReviews x91 x92 x93 x94)
then show ?thesis using * Ract by (clarsimp; metis Suc-leD eqButPID-imp2
not-less-eq-eq s's1')
qed auto

```

```

next
  case (Lact x5)
  then show ?thesis using * by (cases x5; auto)
qed
qed

```

**definition**  $\Delta 1 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 1\ s\ vl\ s1\ vl1 \equiv \neg (\exists cid. PID \in \in paperIDs\ s\ cid) \wedge s = s1 \wedge B\ vl\ vl1$

**definition**  $\Delta 2 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 2\ s\ vl\ s1\ vl1 \equiv$   
 $\exists cid. PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid = subPH \wedge eqButPID\ s\ s1$

**definition**  $\Delta 3 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 3\ s\ vl\ s1\ vl1 \equiv$   
 $\exists cid. PID \in \in paperIDs\ s\ cid \wedge eqButPID\ s\ s1 \wedge phase\ s\ cid > subPH \wedge vl = [] \wedge$   
 $vl1 = []$

**definition**  $\Delta e :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta e\ s\ vl\ s1\ vl1 \equiv$   
 $\exists cid. PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > subPH \wedge vl \neq []$

**lemma** *istate- $\Delta 1$* :  
**assumes**  $B: B\ vl\ vl1$   
**shows**  $\Delta 1\ istate\ vl\ istate\ vl1$   
**using**  $B$  **unfolding**  $\Delta 1\text{-def}$   $B\text{-def}$  **istate-def** **by** *auto*

**lemma** *unwind-cont- $\Delta 1$* : *unwind-cont*  $\Delta 1\ \{\Delta 1, \Delta 2, \Delta e\}$   
**proof**(*rule, goal-cases*)  
**case** ( $1\ s\ vl\ s1\ vl1$ )

**assume**  $rsT: reachNT\ s$  **and**  $rs1: reach\ s1$  **and**  $\Delta 1\ s\ vl\ s1\ vl1$   
**hence**  $rs: reach\ s$  **and**  $ss1: s1 = s$  **and**  $vl: vl \neq []$   
**and**  $pid: \forall cid. PID \notin set\ (paperIDs\ s\ cid)$   
**using** *reachNT-reach* **unfolding**  $\Delta 1\text{-def}$   $B\text{-def}$  **by** *auto*  
**show** ?case **(is ?iact  $\vee$  ( $\neg$   $\wedge$  ?react))**  
**proof**–  
**have** ?react **proof** (*rule, goal-cases*)  
**case** ( $1\ a\ ou\ s'\ vl'$ )  
**let** ?trn = *Trans*  $s\ a\ ou\ s'$  **let** ?trn1 = *Trans*  $s1\ a\ ou\ s'$   
**assume** *step*: *step*  $s\ a = (ou, s')$   
**and**  $T: \neg T\ ?trn$  **and**  $c: consume\ ?trn\ vl\ vl'$   
**have**  $\varphi: \neg \varphi\ ?trn$   
**apply**(*cases*  $a$ )  
**subgoal** **by** *simp*  
**subgoal** **for**  $x2$  **apply**(*cases*  $x2$ ) **using** *step*  $pid$  **by**(*auto* *simp*: *u-defs*)  
**by** *simp-all*  
**hence**  $vl': vl' = vl$  **using**  $c$  **unfolding** *consume-def* **by** *auto*  
**show** ?case **(is ?match  $\vee$  ?ignore)**

```

proof–
  have ?match proof
    show validTrans ?trn1 unfolding ss1 using step by simp
  next
    show consume ?trn1 vl1 vl1 unfolding consume-def ss1 using  $\varphi$  by auto
  next
    show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
  next
    assume  $\gamma$  ?trn thus  $g$  ?trn =  $g$  ?trn1 unfolding ss1 by simp
  next
    show disjAll { $\Delta 1$ ,  $\Delta 2$ ,  $\Delta e$ } s' vl' s' vl1
    proof (cases  $\exists$  cid. PID  $\in$  paperIDs s' cid)
      case False hence  $\Delta 1$  s' vl' s' vl1
        by (simp add:  $\Delta 1$ -def B-def vl vl')
      thus ?thesis by simp
    next
      case True
        hence  $\Delta 2$  s' vl' s' vl1
          using step pid
          apply (simp add:  $\Delta 2$ -def vl' vl)
          apply (erule exE)
          subgoal for cid apply (rule exI[of - cid])
            apply (cases a)
              subgoal for x1 apply (cases x1, auto simp: c-defs) [] .
              subgoal for x2 apply (cases x2, auto simp: u-defs) [] .
              subgoal for x3 apply (cases x3, auto simp: uu-defs) [] .
            by simp-all
          done
        thus ?thesis by simp
      qed
    qed
    thus ?thesis by simp
  qed
  qed
  thus ?thesis using vl by simp
qed
qed

```

```

lemma unwind-cont- $\Delta 2$ : unwind-cont  $\Delta 2$  { $\Delta 2, \Delta 3, \Delta e$ }
proof(rule,goal-cases)
  case (1 s vl s1 vl1)
  assume rsT: reachNT s and rs1: reach s1 and  $\Delta 2$  s vl s1 vl1
  then obtain cid where rs: reach s
    and pid: PID  $\in$  paperIDs s cid and ss1: eqButPID s s1
    and ph: phase s cid = subPH
    using reachNT-reach unfolding  $\Delta 2$ -def by auto

  have cid: cid  $\in$  confIDs s
    by (metis paperIDs-confIDs pid rs)

```

```

from pid ph cid have
  pid1:  $PID \in \in paperIDs\ s1\ cid$ 
  and ph1:  $phase\ s1\ cid = subPH$ 
  and cid1:  $cid \in \in confIDs\ s1$ 
  by (auto simp add: eqButPID-imp[OF ss1])

show ?case (is ?iact  $\vee$  ( $\neg$   $\wedge$  ?react))
proof (cases vl1)
  case (Cons v vl1') note this[simp]
  obtain uid1 where aut1:  $isAut\ s1\ cid\ uid1\ PID$ 
  thm paperID-ex-userID
  using paperID-ex-userID[OF rs1 pid1] by auto
  have uid1:  $uid1 \in \in userIDs\ s1$ 
  by (metis roles-userIDs rs1 aut1)

from aut1 have  $isAut\ s\ cid\ uid1\ PID$ 
  using ss1 aut1 by (simp add: eqButPID-imp[OF ss1])
with reachNT-non-isAut-isPC-isChair[OF rsT] uid1 have uid1-ne:  $uid1 \notin UIDs$ 
  by auto

let ?a1 = (Uact (uPaperC cid uid1 (pass s1 uid1) PID v))
obtain s1' where step:  $step\ s1\ ?a1 = (outOK, s1')$  and s1's1:  $eqButPID\ s1'\ s1$ 
by (cases paper s1 PID)
  (auto simp add: u-defs cid1 uid1 pid1 ph1 aut1 eqButPID-def eqButPID-def)

have ?iact
proof
  show  $step\ s1\ ?a1 = (outOK, s1')$  using step .
  show  $\varphi (Trans\ s1\ ?a1\ outOK\ s1')$  by simp
  show consume ( $Trans\ s1\ ?a1\ outOK\ s1'$ ) vl1 vl1' by (simp add: consume-def)
  show  $\neg \gamma (Trans\ s1\ ?a1\ outOK\ s1')$  by (simp add: uid1-ne)
  have  $\Delta 2\ s\ vl\ s1'\ vl1'$  unfolding  $\Delta 2$ -def
    apply (rule exI[where x=cid])
    using ph pid
    apply clarsimp
    by (metis s1's1 eqButPID-sym eqButPID-trans ss1)
  thus disjAll  $\{\Delta 2, \Delta 3, \Delta e\}$  s vl s1' vl1' by simp
qed
thus ?thesis by simp
next
case Nil note this[simp]
have ?react
proof (rule, goal-cases)
  case (1 a ou s' vl')
  assume STEP:  $step\ s\ a = (ou, s')$  and  $\neg T (Trans\ s\ a\ ou\ s')$ 
  and CONSUME: consume ( $Trans\ s\ a\ ou\ s'$ ) vl vl'

```

```

have ph': phase s' cid  $\geq$  subPH
  by (smt (verit) STEP ph phase-increases)

have pid': PID  $\in\in$  paperIDs s' cid using pid STEP
  by (metis paperIDs-mono)

{
  fix s1 vl1
  assume phase s' cid  $\neq$  subPH vl'≠[]
  hence  $\Delta e$  s' vl' s1 vl1
    unfolding  $\Delta e$ -def
    apply –
    apply(rule exI[of - cid])
    using STEP CONSUME ph
    apply (cases a)
    subgoal for x1 apply (cases x1) apply(auto simp: c-defs) .
    subgoal for x2 apply (cases x2) apply(auto simp: u-defs consume-def
pid) .
    subgoal for x3 apply (cases x3) apply(auto simp: uu-defs) .
    by simp-all
  } note  $\Delta e$ =this

obtain s1' ou' where
  STEP': step s1 a = (ou',s1') and s's1': eqButPID s' s1'
  using eqButPID-step[OF ss1 STEP]
  by fastforce

from eqButPID-step-φ[OF ss1 STEP STEP']
have  $\varphi$ -eq:  $\varphi$  (Trans s1 a ou' s1') =  $\varphi$  (Trans s a ou s') by simp

show ?case (is ?match  $\vee$  ?ignore)
proof (cases  $\varphi$  (Trans s a ou s'))
  case True note  $\varphi$ =this

then obtain cid' uid p where
  a[simp]: a=Uact (uPaperC cid' uid p PID (hd vl)) ou=outOK
  using CONSUME
  by (cases (Trans s a ou s') rule: f.cases) (auto simp add: consume-def)

from STEP pid have [simp]: cid'=cid
  by (simp add: u-defs paperIDs-equals[OF rs])

from  $\varphi$ -step-eqButPID[OF φ STEP] have ss': eqButPID s s' .

have nγ:  $\neg\gamma$  (Trans s a ou s')
  using P-φ-γ[OF rsT STEP] by simp

have ph': phase s' cid = subPH

```

```

using STEP by (auto simp add: u-defs)

have ?ignore
proof
show  $\neg \gamma$  (Trans s a ou s') by (rule n $\gamma$ )
have  $\Delta 2$  s' vl' s1 vl1
  unfolding  $\Delta 2$ -def
  using ph' pid' eqButPID-trans[OF eqButPID-sym[OF ss']] ss1
  by auto
thus disjAll { $\Delta 2$ ,  $\Delta 3$ ,  $\Delta e$ } s' vl' s1 vl1 by simp
qed
thus ?thesis by simp
next
case False note  $\varphi$ =this
with CONSUME have [simp]: vl'=vl by (simp add: consume-def)

have ?match proof
show validTrans (Trans s1 a ou' s1') using STEP' by simp
show consume (Trans s1 a ou' s1') vl1 vl1 using  $\varphi$ 
  by (simp add: consume-def  $\varphi$ -eq)
show  $\gamma$  (Trans s a ou s') =  $\gamma$  (Trans s1 a ou' s1') by simp
show  $\gamma$  (Trans s a ou s')  $\implies$  g (Trans s a ou s') = g (Trans s1 a ou' s1')
  using eqButPID-step-out[OF ss1 STEP STEP' rsT rs1 pid]
  by simp
show disjAll { $\Delta 2$ ,  $\Delta 3$ ,  $\Delta e$ } s' vl' s1' vl1
proof (cases phase s' cid = subPH)
case True
  hence  $\Delta 2$  s' vl' s1' vl1
  unfolding  $\Delta 2$ -def
  using eqButPID-step[OF ss1 STEP STEP']
  using ph' pid' by auto
  thus ?thesis by simp
next
case False with ph' have ph': subPH < phase s' cid by simp
show ?thesis proof (cases vl'=[])
case False
  hence  $\Delta e$  s' vl' s1' vl1 using  $\Delta e$  ph' by simp
  thus ?thesis by simp
next
case True
  hence  $\Delta 3$  s' vl' s1' vl1
  unfolding  $\Delta 3$ -def
  apply(intro exI[of - cid])
  using ph' pid' eqButPID-step[OF ss1 STEP STEP']
  by simp
  thus ?thesis by simp
qed
qed
qed

```

thus ?thesis by simp  
 qed  
 qed  
 thus ?thesis by simp  
 qed  
 qed

**lemma** *unwind-cont- $\Delta\mathcal{B}$* : *unwind-cont  $\Delta\mathcal{B}$   $\{\Delta\mathcal{B},\Delta e\}$*

**proof** (*rule, goal-cases*)

**case** (*1 s vl s1 vl1*)

**assume** *rsT: reachNT s and rs1: reach s1 and  $\Delta$ :  $\Delta\mathcal{B}$  s vl s1 vl1*

**thm**  *$\Delta\mathcal{B}$ -def*

**then obtain** *cid where ss1: eqButPID s s1 and [simp]: vl=[] vl1=[]*

**and** *pid: PID  $\in\in$  paperIDs s cid and ph: subPH < phase s cid*

**unfolding**  *$\Delta\mathcal{B}$ -def by auto*

**from** *rsT have rs: reach s*

**by** (*metis reachNT-reach*)

**from** *pid ph have*

*pid1: PID  $\in\in$  paperIDs s1 cid*

**and** *ph1: subPH < phase s1 cid*

**using** *ss1*

**by** (*auto simp add: eqButPID-imp*)

**thus** ?case (**is** -  $\vee$  (-  $\wedge$  ?react))

**proof** -

**have** ?react

**proof** (*rule, goal-cases*)

**case** (*1 a ou s' vl'*)

**assume** *STEP: step s a = (ou, s') and NT:  $\neg T$  (Trans s a ou s') (is  $\neg T$  ?trn)*

**and** *CONSUME: consume (Trans s a ou s') vl vl'*

**show** ?case (**is** ?match  $\vee$  -)

**proof** -

**have** *ph': subPH < phase s' cid*

**using** *STEP ph phase-increases by (meson le-trans not-less)*

**have** [simp]: *vl'=[] using CONSUME by (auto simp add: consume-def)*

**obtain** *ou1 and s1' where STEP1: step s1 a = (ou1, s1')*

**by** (*metis prod.exhaust*)

**let** ?trn1 = *Trans s1 a ou1 s1'*

**have** *s's1': eqButPID s' s1' using eqButPID-step[OF ss1 STEP STEP1] .*

**from** *s's1' ph' have ph1': subPH < phase s1' cid*

```

    by (simp add: eqButPID-imp)

  have  $\varphi: \neg \varphi ?trn1$ 
  using STEP1 ph1' unfolding  $\varphi$ -def2 by (auto simp: u-defs paperIDs-equals[OF
rs1 pid1])

  have ?match proof
    show validTrans ?trn1 using STEP1 by simp
  next
    show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi$  by auto
  next
    show  $\gamma ?trn = \gamma ?trn1$  unfolding ss1 by simp
  next
    assume  $\gamma ?trn$  thus  $g ?trn = g ?trn1$ 
    using eqButPID-step-out[OF ss1 STEP STEP1 rsT rs1 pid] by simp
  next
    have  $\Delta 3 s' vl' s1' vl1$  using ph' s's1' paperIDs-mono[OF STEP pid]
    unfolding  $\Delta 3$ -def by auto
    thus disjAll  $\{\Delta 3, \Delta e\} s' vl' s1' vl1$  by simp
  qed
  thus ?thesis by simp
qed
qed
thus ?case by simp
qed
qed

```

**definition** *K1exit* where

$K1exit\ s \equiv \exists cid. phase\ s\ cid > subPH \wedge PID \in \in paperIDs\ s\ cid$

**lemma** *invarNT-K1exit*: *invarNT K1exit*

```

  unfolding invarNT-def apply (safe dest!: reachNT-reach)
  subgoal for - a apply (cases a)
    subgoal for x1 apply (cases x1, auto simp: c-defs K1exit-def) .
    subgoal for x2 apply (cases x2)
      apply (auto simp: u-defs K1exit-def paperIDs-equals)
      apply (metis less-nat-zero-code)
      apply (metis Suc-lessD) .
    subgoal for x3 apply (cases x3, auto simp: uu-defs K1exit-def) .
  by simp-all
done

```

**lemma** *noVal-K1exit*: *noVal K1exit v*

```

  apply (rule no $\varphi$ -noVal)
  unfolding no $\varphi$ -def apply safe
  subgoal for - a apply (cases a)
    subgoal by simp
    subgoal for x2

```

```

    apply(cases x2, auto simp add: u-defs K1exit-def) []
    apply (metis reachNT-reach less-not-refl paperIDs-equals) .
  by simp-all
done

```

**lemma** *unwind-exit- $\Delta e$ : unwind-exit  $\Delta e$*

**proof**

```

  fix s s1 :: state and vl vl1 :: value list
  assume rsT: reachNT s and rs1: reach s1 and  $\Delta e$ :  $\Delta e$  s vl s1 vl1
  hence vl: vl  $\neq$  [] using reachNT-reach unfolding  $\Delta e$ -def by auto
  hence K1exit s using  $\Delta e$  unfolding K1exit-def  $\Delta e$ -def by auto
  thus vl  $\neq$  []  $\wedge$  exit s (hd vl) apply(simp add: vl)
  by (metis rsT exitI2 invarNT-K1exit noVal-K1exit)
qed

```

**theorem** *secure: secure*

```

  apply(rule unwind-decomp3-secure[of  $\Delta 1$   $\Delta 2$   $\Delta e$   $\Delta 3$ ])
  using
    istate- $\Delta 1$ 
    unwind-cont- $\Delta 1$  unwind-cont- $\Delta 2$  unwind-cont- $\Delta 3$ 
    unwind-exit- $\Delta e$ 
  by auto

```

**end**

**theory** *Paper-Aut*

**imports** *../Observation-Setup Paper-Value-Setup Bounded-Deducibility-Security.Compositional-Reasoning*  
**begin**

## 5.4 Confidentiality protection from non-authors

We verify the following property:

A group of users UIDs learns nothing about the various uploads of a paper PID except for the last (most recent) upload unless/until a user in UIDs becomes an author of the paper.

```

fun T :: (state,act,out) trans  $\Rightarrow$  bool where
  T (Trans - - ou s') = ( $\exists$  uid  $\in$  UIDs. isAUT s' uid PID)

```

**declare** *T.simps* [*simp del*]

```

definition B :: value list  $\Rightarrow$  value list  $\Rightarrow$  bool where
  B vl vl1  $\equiv$  vl  $\neq$  []  $\wedge$  vl1  $\neq$  []  $\wedge$  last vl = last vl1

```

**interpretation** *BD-Security-IO* **where**

```

  istate = istate and step = step and
   $\varphi$  =  $\varphi$  and f = f and  $\gamma$  =  $\gamma$  and g = g and T = T and B = B

```

**done**

**lemma** *reachNT-non-isAut*:

**assumes** *reachNT s* **and** *uid*  $\in$  *UIDs*

**shows**  $\neg$  *isAut s cid uid PID*

**using** *assms*

**apply** *induct*

**apply** (*auto simp: istate-def*)[]

**subgoal for** *trn* **apply**(*cases trn, auto simp: T.simps reachNT-reach isChair-isPC isAUT-def*) .

**done**

**lemma** *T- $\varphi$ - $\gamma$* :

**assumes** *1: reachNT s* **and** *2: step s a = (ou,s<sup>^</sup>)  $\varphi$  (Trans s a ou s<sup>^</sup>)*

**shows**  $\neg$   $\gamma$  (*Trans s a ou s<sup>^</sup>*)

**using** *reachNT-non-isAut[OF 1] 2 unfolding T.simps  $\varphi$ -def2*

**by** (*auto simp add: u-defs*)

**lemma** *eqButPID-step-out*:

**assumes** *ss1: eqButPID s s1*

**and** *step: step s a = (ou,s<sup>^</sup>)* **and** *step1: step s1 a = (ou1,s1<sup>^</sup>)*

**and** *sT: reachNT s* **and** *s1: reach s1*

**and** *PID: PID  $\in$  paperIDs s cid*

**and** *ph: phase s cid = subPH*

**and** *UIDs: userOfA a  $\in$  UIDs*

**shows** *ou = ou1*

**proof**–

**note** *Inv = reachNT-non-isAut[OF sT UIDs]*

**note** *eqs = eqButPID-imp[OF ss1]*

**note** *eqs' = eqButPID-imp1[OF ss1]*

**note** *s = reachNT-reach[OF sT]*

**have** *PID': PID  $\in$  paperIDs s1 cid* **and** *ph': phase s cid = subPH*

**using** *PID ph ss1 unfolding eqButPID-def by auto*

**note** *simps[simp] = c-defs u-defs uu-defs r-defs l-defs Paper-dest-conv eqButPID-def eqButPID-def eqButC*

**note** *\* = step step1 eqs eqs' s s1 PID ph PID' ph' UIDs paperIDs-equals[OF s] Inv*

**show** *?thesis*

**proof** (*cases a*)

**case** (*Cact x1*)

**then show** *?thesis using \* by (cases x1; auto)*

**next**

**case** (*Uact x2*)

**then show** *?thesis using \* by (cases x2; auto)*

**next**

**case** (*UUact x3*)

```

    then show ?thesis using * by (cases x3; auto)
next
  case (Ract x4)
  with * show ?thesis
  proof (cases x4)
    case (rPaperC x61 x62 x63 x64)
    then show ?thesis using * Ract by (clarsimp; metis Suc-n-not-le-n)
  next
    case (rMyReview x81 x82 x83 x84)
    then show ?thesis using * Ract by (auto simp: getNthReview-def)
  next
    case (rReviews x91 x92 x93 x94)
    then show ?thesis using * Ract by (clarsimp; metis Suc-leD eqButPID-imp2
not-less-eq-eq ss1)
  qed auto
next
  case (Lact x5)
  then show ?thesis using * by (cases x5; auto)
qed
qed

```

major

```

lemma eqButPID-step-eq:
assumes ss1: eqButPID s s1
and [simp]: a=Uact (uPaperC cid wid p PID ct) ou=outOK
and step: step s a = (ou, s') and step1: step s1 a = (ou', s1')
shows s' = s1'
  using ss1 step step1
  apply (simp add: u-defs eqButPID-paper )
  subgoal by (cases s; cases s1; auto simp add: eqButPID-def eeqButPID-def)
  subgoal by (use ss1 step step1 in ⟨auto simp add: eqButPID-def eeqButPID-def⟩)
  done

```

```

definition Δ1 :: state ⇒ value list ⇒ state ⇒ value list ⇒ bool where
Δ1 s vl s1 vl1 ≡
¬ (∃ cid. PID ∈∈ paperIDs s cid) ∧ s = s1 ∧ B vl vl1

```

```

definition Δ2 :: state ⇒ value list ⇒ state ⇒ value list ⇒ bool where
Δ2 s vl s1 vl1 ≡
(∃ cid. PID ∈∈ paperIDs s cid ∧ phase s cid = subPH) ∧
eqButPID s s1 ∧ B vl vl1

```

```

definition Δ3 :: state ⇒ value list ⇒ state ⇒ value list ⇒ bool where
Δ3 s vl s1 vl1 ≡
(∃ cid. PID ∈∈ paperIDs s cid) ∧ s = s1 ∧ vl = [] ∧ vl1 = []

```

```

definition Δe :: state ⇒ value list ⇒ state ⇒ value list ⇒ bool where

```

$\Delta e s vl s1 vl1 \equiv$   
 $(\exists cid. PID \in \in paperIDs s cid \wedge phase s cid > subPH) \wedge vl \neq []$

**lemma** *istate- $\Delta 1$* :  
**assumes**  $B: B vl vl1$   
**shows**  $\Delta 1 istate vl istate vl1$   
**using** *assms unfolding  $\Delta 1$ -def  $B$ -def *istate-def* by auto*

**lemma** *unwind-cont- $\Delta 1$* : *unwind-cont  $\Delta 1 \{\Delta 1, \Delta 2, \Delta e\}$*

**proof**(*rule*)

**let**  $?\Delta = disjAll \{\Delta 1, \Delta 2, \Delta e\}$

**fix**  $s s1 :: state$  **and**  $vl vl1 :: value list$

**assume**  $rsT: reachNT s$  **and**  $rs1: reach s1$  **and**  $\Delta 1 s vl s1 vl1$

**hence**  $rs: reach s$  **and**  $cid: \forall cid. \neg PID \in \in paperIDs s cid$

**and**  $ss1: s1 = s$  **and**  $vl: vl \neq []$  **and**  $vl1: vl1 \neq []$  **and**  $vlvl1: last vl = last vl1$

**using** *reachNT-reach unfolding  $\Delta 1$ -def  $B$ -def* by auto

**show** *iaction  $?\Delta s vl s1 vl1 \vee$*

$((vl \neq [] \vee vl1 = []) \wedge reaction ?\Delta s vl s1 vl1)$  (**is**  $?iact \vee (- \wedge ?react)$ )

**proof** –

**have**  $?react$

**proof** (*rule, goal-cases*)

**case** ( $1 a ou s' vl'$ )

**assume**  $STET: step s a = (ou, s')$  **and**  $\neg T (Trans s a ou s')$

**and**  $CONSUME: consume (Trans s a ou s') vl vl'$

**have** *not-phi*:  $\neg \varphi (Trans s a ou s')$

**using**  $STET cid$

**by** (*auto simp:  $\varphi$ -def2 u-defs*)

**with**  $CONSUME$  **have**  $vlvl': vl'=vl$

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

**have** *match* ( $disjAll \{\Delta 1, \Delta 2, \Delta e\}$ )  $s s1 vl1 a ou s' vl'$

**proof**

**show** *validTrans* ( $Trans s1 a ou s'$ ) **using**  $STET$  **by** (*simp add: ss1*)

**show** *consume* ( $Trans s1 a ou s'$ )  $vl1 vl1$

**by** (*simp add: consume-def ss1 not-phi*)

**show**  $\gamma (Trans s a ou s') = \gamma (Trans s1 a ou s')$  **by** *simp*

**show**  $g (Trans s a ou s') = g (Trans s1 a ou s')$  **by** *simp*

**show**  $disjAll \{\Delta 1, \Delta 2, \Delta e\} s' vl' s' vl1$

**proof** (*cases  $\exists cid. PID \in \in paperIDs s' cid$* )

**case** *False* **hence**  $\Delta 1 s' vl' s' vl1$

**by** (*simp add:  $\Delta 1$ -def  $B$ -def  $vlvl' vl vl1 vlvl1$* )

**thus** *?thesis* **by** *simp*

**next**

**case** *True* **hence**  $\Delta 2 s' vl' s' vl1$

**apply** (*simp add:  $\Delta 2$ -def  $B$ -def  $vlvl' vl vl1 vlvl1$* )

**apply** (*erule exE*)

**subgoal for**  $cid$  **apply**(*rule exI[of - cid]*)

```

    apply simp
    apply (use STET cid in ⟨cases a⟩)
    subgoal for x1 apply(cases x1) apply(auto simp: c-defs) .
    subgoal for x2 apply(cases x2) apply(auto simp: u-defs) .
    subgoal for x3 apply(cases x3) apply(auto simp: uu-defs) .
    subgoal by simp
    subgoal by simp
    done
  done
  thus ?thesis by simp
qed
qed
thus ?case by simp
qed
thus ?thesis using vl vl1 by auto
qed
qed

```

**lemma** *unwind-cont- $\Delta 2$* : *unwind-cont  $\Delta 2$   $\{\Delta 2, \Delta 3, \Delta e\}$*

**proof** (*rule, goal-cases*)

**case** (*1 s vl s1 vl1*)

**assume** *rsT: reachNT s and rs1: reach s1 and  $\Delta 2$  s vl s1 vl1*

**then obtain** *cid* **where**

*rs: reach s and pid: PID  $\in \in$  paperIDs s cid and ph: phase s cid = subPH*

**and** *ss1: eqButPID s s1*

**and** *vl: vl  $\neq$  [] and vl1: vl1  $\neq$  [] and vlvl1: last vl = last vl1*

**using** *reachNT-reach unfolding  $\Delta 2$ -def B-def* **by** *auto*

**have** *cid: cid  $\in \in$  confIDs s*

**by** (*metis paperIDs-confIDs pid rs*)

**from** *pid ph cid* **have**

*pid1: PID  $\in \in$  paperIDs s1 cid*

**and** *ph1: phase s1 cid = subPH*

**and** *cid1: cid  $\in \in$  confIDs s1*

**by** (*auto simp add: eqButPID-imp[OF ss1]*)

**show** *?case (is ?iact  $\vee$  ( $- \wedge$  ?react))*

**proof** (*cases length vl1 > 1*)

**case** *True* **then obtain** *v vl1'* **where** [*simp*]: *vl1 = v#vl1' vl1'  $\neq$  []* **by** (*cases vl1*) *auto*

**obtain** *uid1* **where** *aut1: isAut s1 cid uid1 PID*

**thm** *paperID-ex-userID*

**using** *paperID-ex-userID[OF rs1 pid1]* **by** *auto*

**have** *uid1: uid1  $\in \in$  userIDs s1*

**by** (*metis roles-userIDs rs1 aut1*)

**from** *aut1* **have** *isAut s cid uid1 PID*

```

    using ss1 aut1 by (simp add: eqButPID-imp[OF ss1])
with reachNT-non-isAut[OF rsT] uid1 have uid1-ne: uid1 ∉ UIDs
  by auto

let ?a1 = (Uact (uPaperC cid uid1 (pass s1 uid1) PID v))
obtain s1' where step: step s1 ?a1 = (outOK,s1') and s1's1: eqButPID s1'
s1
  apply (simp add: u-defs cid1 uid1 pid1 ph1 aut1)
  apply (cases paper s1 PID)
  apply (auto simp: eqButPID-def eqButPID-def)
  done

have ?iact
proof
  show step s1 ?a1 = (outOK,s1') using step .
  show  $\varphi$  (Trans s1 ?a1 outOK s1') by simp
  show consume (Trans s1 ?a1 outOK s1') vl vl1' by (simp add: consume-def)
  show  $\neg \gamma$  (Trans s1 ?a1 outOK s1') by (simp add: uid1-ne)
  have  $\Delta 2$  s vl s1' vl1' unfolding  $\Delta 2$ -def B-def
    using vl vl1 ph pid
    apply simp-all
    by (metis s1's1 eqButPID-sym eqButPID-trans ss1)
  thus disjAll { $\Delta 2$ ,  $\Delta 3$ ,  $\Delta e$ } s vl s1' vl1' by simp
qed
thus ?thesis by simp
next
case False then obtain v1 where [simp]: vl1=[v1] using vl1 by (cases vl1)
auto

have ?react
proof (rule, goal-cases)
  case (1 a ou s' vl')
  assume STET: step s a = (ou, s') and  $\neg T$  (Trans s a ou s')
  and CONSUME: consume (Trans s a ou s') vl vl'

  have ph': phase s' cid  $\geq$  subPH
  by (smt (verit) STET ph phase-increases)

  have pid': PID  $\in \in$  paperIDs s' cid using pid STET
  by (metis paperIDs-mono)

  {
  fix s1 vl1
  assume phase s' cid  $\neq$  subPH
  hence  $\Delta e$  s' vl' s1 vl1
  unfolding  $\Delta e$ -def
  using STET CONSUME vl ph
  apply (cases a)
  subgoal for x1 apply(cases x1) apply(auto simp: c-defs) .
  }

```

```

      subgoal for x2 apply(cases x2) apply(auto simp: u-defs consume-def
pid) apply metis .
      subgoal for x3 apply(cases x3) apply(auto simp: uu-defs) .
      by simp-all
    } note Δe=this

  obtain s1' ou' where STET': step s1 a = (ou',s1') and s's1': eqButPID s'
s1'
    using eqButPID-step[OF ss1 STET]
    by fastforce

  from eqButPID-step-φ[OF ss1 STET STET']
  have φ-eq: φ (Trans s1 a ou' s1') = φ (Trans s a ou s') by simp

  show ?case (is ?match ∨ ?ignore)
  proof (cases φ (Trans s a ou s'))
    case True note φ=this

      then obtain cid' uid p where
        a[simp]: a=Uact (uPaperC cid' uid p PID (hd vl)) ou=outOK using
CONSUME
        by (cases (Trans s a ou s') rule: f.cases) (auto simp add: consume-def vl)

      from STET pid have [simp]: cid'=cid
        by (simp add: u-defs paperIDs-equals[OF rs])

      from φ-step-eqButPID[OF φ STET] have ss': eqButPID s s' .

      have nγ: ¬γ (Trans s a ou s')
        using T-φ-γ[OF rsT STET] by simp

      have ph': phase s' cid = subPH
        using STET by (auto simp add: u-defs)

      show ?thesis proof (cases length vl = 1)
        case True hence [simp]: vl=[v1] using vl1 by (cases vl) simp-all
        from CONSUME have [simp]: vl'=[] by (simp add: consume-def φ)

        from STET STET' have [simp]: s1'=s'
          using eqButPID-step-eq ss1 a by blast

        have ?match proof
          show validTrans (Trans s1 a ou' s1') using STET' by simp
          show consume (Trans s1 a ou' s1') vl1 []
            using φ φ-eq CONSUME
            by (simp add: consume-def)
          show γ (Trans s a ou s') = γ (Trans s1 a ou' s1') by simp
          show γ (Trans s a ou s') ⇒ g (Trans s a ou s') = g (Trans s1 a ou'
s1')

```

```

    using n $\gamma$  by simp
    have  $\Delta 3$  s' vl' s1' []
    unfolding  $\Delta 3$ -def
    using ph' pid'
    by force
    thus disjAll { $\Delta 2$ ,  $\Delta 3$ ,  $\Delta e$ } s' vl' s1' [] by simp
qed
thus ?thesis by simp
next
case False then obtain v where [simp]: vl=v#vl' vl'≠[]
    using CONSUME vl by (cases vl) (simp-all add: consume-def)
    have ?ignore
    proof
    show  $\neg \gamma$  (Trans s a ou s') by (rule n $\gamma$ )
    have  $\Delta 2$  s' vl' s1 vl1
    unfolding  $\Delta 2$ -def B-def
    using vlvl1 ph' pid' eqButPID-trans[OF eqButPID-sym[OF ss']] ss1]
    by auto
    thus disjAll { $\Delta 2$ ,  $\Delta 3$ ,  $\Delta e$ } s' vl' s1 vl1 by simp
    qed
    thus ?thesis by simp
    qed
next
case False note  $\varphi$ =this
with CONSUME have [simp]: vl'=vl by (simp add: consume-def)

have ?match proof
    show validTrans (Trans s1 a ou' s1') using STET' by simp
    show consume (Trans s1 a ou' s1') vl1 vl1 using  $\varphi$ 
    by (simp add: consume-def  $\varphi$ -eq)
    show  $\gamma$  (Trans s a ou s') =  $\gamma$  (Trans s1 a ou' s1') by simp
    show  $\gamma$  (Trans s a ou s')  $\implies$  g (Trans s a ou s') = g (Trans s1 a ou' s1')
    using eqButPID-step-out[OF ss1 STET STET' rsT rs1 pid ph]
    by simp
    show disjAll { $\Delta 2$ ,  $\Delta 3$ ,  $\Delta e$ } s' vl' s1' vl1
    proof (cases phase s' cid = subPH)
    case True
    hence  $\Delta 2$  s' vl' s1' vl1
    unfolding  $\Delta 2$ -def B-def
    using eqButPID-step[OF ss1 STET STET']
    using ph' pid' vl vl1 vlvl1 by auto
    thus ?thesis by simp
    next
    case False
    hence  $\Delta e$  s' vl' s1' vl1 using  $\Delta e$  by simp
    thus ?thesis by simp
    qed
    qed
thus ?thesis by simp

```

```

    qed
  qed
  thus ?thesis using vl vl1 by simp
  qed
  qed
  lemma unwind-cont- $\Delta\beta$ : unwind-cont  $\Delta\beta$  { $\Delta\beta, \Delta e$ }
  proof (rule, goal-cases)
    case (1 s vl s1 vl1)
    assume rsT: reachNT s and rs1: reach s1 and  $\Delta$ :  $\Delta\beta$  s vl s1 vl1
    thm  $\Delta\beta$ -def
    then obtain cid where [simp]: s1=s vl=[] vl1=[]
      and pid: PID  $\in\in$  paperIDs s cid
      unfolding  $\Delta\beta$ -def by auto

    thus ?case (is -  $\vee$  (-  $\wedge$  ?react))
    proof -
      have ?react
      proof (rule, goal-cases)
        case (1 a ou s' vl')
        assume STET: step s a = (ou, s') and NT:  $\neg$  T (Trans s a ou s')
          and CONSUME: consume (Trans s a ou s') vl vl'
        have  $\Delta\beta$ :  $\Delta\beta$  s' vl' s' vl' and [simp]: vl'=[]
          using CONSUME paperIDs-mono[OF STET pid]
          unfolding  $\Delta\beta$ -def
          by (auto simp add: consume-def)
        thus ?case (is ?match  $\vee$  ?ignore)
        proof -
          have ?match
            apply (rule matchI[of s1 a ou s' vl1 vl1])
            using STET CONSUME  $\Delta\beta$  by simp-all
          thus ?thesis by simp
        qed
      qed
    thus ?thesis by simp
  qed
  qed
  definition K1exit where
  K1exit s  $\equiv$   $\exists$  cid. phase s cid > subPH  $\wedge$  PID  $\in\in$  paperIDs s cid

  lemma invarNT-K1exit: invarNT K1exit
  unfolding invarNT-def
  apply (safe dest!: reachNT-reach)
  subgoal for - a apply(cases a)
    subgoal for x1 apply(cases x1) apply (auto simp: c-defs K1exit-def) .
    subgoal for x2 apply(cases x2) apply (auto simp: u-defs K1exit-def pa-
  perIDs-equals,force+) .
    subgoal for x3 apply(cases x3) apply (auto simp: uu-defs K1exit-def) .

```

```

    by simp-all
  done

lemma noVal-K1exit: noVal K1exit v
  apply(rule no $\varphi$ -noVal)
  unfolding no $\varphi$ -def apply safe
  subgoal for - a apply(cases a)
    subgoal by simp
    subgoal for x2
      apply(cases x2, auto simp add: u-defs K1exit-def) []
      apply (metis reachNT-reach less-not-refl paperIDs-equals) .
    by simp-all
  done

lemma unwind-exit- $\Delta e$ : unwind-exit  $\Delta e$ 
proof
  fix s s1 :: state and vl vl1 :: value list
  assume rsT: reachNT s and rs1: reach s1 and  $\Delta e$ :  $\Delta e$  s vl s1 vl1
  hence vl: vl  $\neq$  [] using reachNT-reach unfolding  $\Delta e$ -def by auto
  hence K1exit s using  $\Delta e$  unfolding K1exit-def  $\Delta e$ -def by auto
  thus vl  $\neq$  []  $\wedge$  exit s (hd vl) apply(simp add: vl)
  by (metis rsT exitI2 invarNT-K1exit noVal-K1exit)
qed

theorem secure: secure
  apply(rule unwind-decomp3-secure[of  $\Delta 1$   $\Delta 2$   $\Delta e$   $\Delta 3$ ])
  using
    istate- $\Delta 1$ 
    unwind-cont- $\Delta 1$  unwind-cont- $\Delta 2$  unwind-cont- $\Delta 3$ 
    unwind-exit- $\Delta e$ 
  by auto

end
theory Paper-All
imports
  Paper-Aut-PC
  Paper-Aut
begin

end
theory Review-Intro
imports ../Safety-Properties
begin

```

## 6 Review Confidentiality

In this section, we prove confidentiality properties for the reviews of papers submitted to a conference. The secrets (values) of interest are therefore the

different versions of a given review for a given paper, identified as the  $N$ 'th review of the paper with id  $PID$ .

Here, we have three points of compromise between the bound and the trigger (which yield three properties). Let

- T1 denote “review authorship”
- T2 denote “PC membership having no conflict with that paper and the conference having moved to the discussion phase”
- T3 denote “PC membership or authorship and the conference having moved to the notification phase”

The three bound-trigger combinations are:

- weak trigger (T1 or T2 or T3) paired with strong bound (allowing to learn almost nothing)
- medium trigger (T1 or T2) paired with medium bound (allowing to learn the last edited version before notification)
- strong trigger (T1) paired with weak bound (allowing to learn the last edited version before discussion and all the later versions)

**end**

```
theory Review-Value-Setup
imports Review-Intro
begin
```

```
consts PID :: paperID consts N :: nat
```

( $PID$ ,  $N$ ) identifies uniquely the review under scrutiny

## 6.1 Preliminaries

```
declare updates-commute-paper[simp]
```

Auxiliary definitions:

```
definition eqExcNth where
eqExcNth xs ys n  $\equiv$ 
length xs = length ys  $\wedge$  ( $\forall$   $i < \text{length } xs$ .  $i \neq n \longrightarrow xs!i = ys!i$ )
```

```
lemma eqExcNth-eq[simp,intro!]: eqExcNth xs xs n
unfolding eqExcNth-def by auto
```

```
lemma eqExcNth-sym:
assumes eqExcNth xs xs1 n
```

**shows**  $eqExcNth\ xs1\ xs\ n$   
**using** *assms* **unfolding** *eqExcNth-def* **by** *auto*

**lemma** *eqExcNth-trans*:  
**assumes**  $eqExcNth\ xs\ xs1\ n$  **and**  $eqExcNth\ xs1\ xs2\ n$   
**shows**  $eqExcNth\ xs\ xs2\ n$   
**using** *assms* **unfolding** *eqExcNth-def* **by** *auto*

**fun** *eqExcD* ::  $paper \Rightarrow paper \Rightarrow bool$  **where**  
*eqExcD* (*Paper name info ct reviews dis decs*)  
     (*Paper name1 info1 ct1 reviews1 dis1 decs1*) =  
 ( $name = name1 \wedge info = info1 \wedge ct = ct1 \wedge dis = dis1 \wedge decs = decs1 \wedge$   
 $eqExcNth\ reviews\ reviews1\ N$ )

**lemma** *eqExcD*:  
*eqExcD pap pap1* =  
 ( $titlePaper\ pap = titlePaper\ pap1 \wedge abstractPaper\ pap = abstractPaper\ pap1 \wedge$   
 $contentPaper\ pap = contentPaper\ pap1 \wedge$   
 $disPaper\ pap = disPaper\ pap1 \wedge decsPaper\ pap = decsPaper\ pap1 \wedge$   
 $eqExcNth\ (reviewsPaper\ pap)\ (reviewsPaper\ pap1)\ N$ )  
**by**(*cases pap, cases pap1, auto*)

**lemma** *eqExcD-eq[simp,intro!]*:  $eqExcD\ pap\ pap$   
**unfolding** *eqExcD* **using** *eqExcNth-eq* **by** *auto*

**lemma** *eqExcD-sym*:  
**assumes**  $eqExcD\ pap\ pap1$   
**shows**  $eqExcD\ pap1\ pap$   
**using** *assms* **unfolding** *eqExcD* **using** *eqExcNth-sym* **by** *auto*

**lemma** *eqExcD-trans*:  
**assumes**  $eqExcD\ pap\ pap1$  **and**  $eqExcD\ pap1\ pap2$   
**shows**  $eqExcD\ pap\ pap2$   
**using** *assms* **unfolding** *eqExcD* **using** *eqExcNth-trans* **by** *auto*

**definition** *eeqExcPID-N* **where**  
*eeqExcPID-N paps paps1*  $\equiv$   
 $\forall\ pid.\ if\ pid = PID\ then\ eqExcD\ (paps\ pid)\ (paps1\ pid)\ else\ paps\ pid = paps1\ pid$

**lemma** *eeqExcPID-N-eeq[simp,intro!]*:  $eeqExcPID-N\ s\ s$   
**unfolding** *eeqExcPID-N-def* **by** *auto*

**lemma** *eeqExcPID-N-sym*:  
**assumes**  $eeqExcPID-N\ s\ s1$  **shows**  $eeqExcPID-N\ s1\ s$   
**using** *assms* *eqExcD-sym* **unfolding** *eeqExcPID-N-def* **by** *auto*

**lemma** *eeqExcPID-N-trans*:  
**assumes**  $eeqExcPID-N\ s\ s1$  **and**  $eeqExcPID-N\ s1\ s2$  **shows**  $eeqExcPID-N\ s\ s2$

**using** *assms eqExcD-trans* **unfolding** *eqExcPID-N-def* **by** *simp blast*

**lemma** *eqExcPID-N-imp*:

*eqExcPID-N paps paps1*  $\implies$  *eqExcD (paps PID) (paps1 PID)*  
 $\llbracket$ *eqExcPID-N paps paps1; pid  $\neq$  PID* $\rrbracket \implies$  *paps pid = paps1 pid*  
**unfolding** *eqExcPID-N-def* **by** *auto*

**lemma** *eqExcPID-N-cong*:

**assumes** *eqExcPID-N paps paps1*  
**and** *pid = PID*  $\implies$  *eqExcD uu uu1*  
**and** *pid  $\neq$  PID*  $\implies$  *uu = uu1*  
**shows** *eqExcPID-N (paps (pid := uu)) (paps1 (pid := uu1))*  
**using** *assms* **unfolding** *eqExcPID-N-def* **by** *auto*

**lemma** *eqExcPID-N-RDD*:

*eqExcPID-N paps paps1*  $\implies$   
*titlePaper (paps PID) = titlePaper (paps1 PID)  $\wedge$*   
*abstractPaper (paps PID) = abstractPaper (paps1 PID)  $\wedge$*   
*contentPaper (paps PID) = contentPaper (paps1 PID)  $\wedge$*   
*disPaper (paps PID) = disPaper (paps1 PID)  $\wedge$*   
*decsPaper (paps PID) = decsPaper (paps1 PID)*  
**using** *eqExcPID-N-def* **unfolding** *eqExcD* **by** *auto*

The notion of two states being equal everywhere except on the the review  $(N, PID)$ :

**definition** *eqExcPID-N* :: *state*  $\Rightarrow$  *state*  $\Rightarrow$  *bool* **where**

*eqExcPID-N s s1*  $\equiv$   
*confIDs s = confIDs s1  $\wedge$  conf s = conf s1  $\wedge$*   
*userIDs s = userIDs s1  $\wedge$  pass s = pass s1  $\wedge$  user s = user s1  $\wedge$  roles s = roles*  
*s1  $\wedge$*   
*paperIDs s = paperIDs s1*  
 $\wedge$   
*eqExcPID-N (paper s) (paper s1)*  
 $\wedge$   
*pref s = pref s1  $\wedge$*   
*voronkov s = voronkov s1  $\wedge$*   
*news s = news s1  $\wedge$  phase s = phase s1*

**lemma** *eqExcPID-N-eq[simp,intro!]*: *eqExcPID-N s s*

**unfolding** *eqExcPID-N-def* **by** *auto*

**lemma** *eqExcPID-N-sym*:

**assumes** *eqExcPID-N s s1* **shows** *eqExcPID-N s1 s*  
**using** *assms eqExcPID-N-sym* **unfolding** *eqExcPID-N-def* **by** *auto*

**lemma** *eqExcPID-N-trans*:

**assumes** *eqExcPID-N s s1* **and** *eqExcPID-N s1 s2* **shows** *eqExcPID-N s s2*  
**using** *assms eqExcPID-N-trans* **unfolding** *eqExcPID-N-def* **by** *auto*

Implications from  $eqExcPID-N$ , including w.r.t. auxiliary operations:

**lemma**  $eqExcPID-N-imp$ :

$eqExcPID-N s s1 \implies$   
 $confIDs s = confIDs s1 \wedge conf s = conf s1 \wedge$   
 $userIDs s = userIDs s1 \wedge pass s = pass s1 \wedge user s = user s1 \wedge roles s = roles$   
 $s1 \wedge$   
 $paperIDs s = paperIDs s1$   
 $\wedge$   
 $eeqExcPID-N (paper s) (paper s1)$   
 $\wedge$   
 $pref s = pref s1 \wedge$   
 $voronkov s = voronkov s1 \wedge$   
 $news s = news s1 \wedge phase s = phase s1 \wedge$

$getAllPaperIDs s = getAllPaperIDs s1 \wedge$   
 $isRev s cid uid pid = isRev s1 cid uid pid \wedge$   
 $getReviewIndex s cid uid pid = getReviewIndex s1 cid uid pid \wedge$   
 $getRevRole s cid uid pid = getRevRole s1 cid uid pid \wedge$   
 $length (reviewsPaper (paper s pid)) = length (reviewsPaper (paper s1 pid))$   
**unfolding**  $eqExcPID-N-def$   $getAllPaperIDs-def$   
**unfolding**  $isRev-def$   $getReviewIndex-def$   $getRevRole-def$  **apply**  $auto$   
**unfolding**  $eeqExcPID-N-def$   $eqExcD$   $eqExcNth-def$  **by**  $(cases pid = PID) auto$

**lemma**  $eqExcPID-N-imp1$ :

$eqExcPID-N s s1 \implies eqExcD (paper s pid) (paper s1 pid)$   
 $eqExcPID-N s s1 \implies pid \neq PID \vee PID \neq pid \implies$   
 $paper s pid = paper s1 pid \wedge$   
 $getNthReview s pid n = getNthReview s1 pid n$   
**unfolding**  $eqExcPID-N-def$   $eeqExcPID-N-def$   $getNthReview-def$   
**apply**  $auto$  **by**  $(metis eqExcD-eq)$

**lemma**  $eqExcPID-N-imp2$ :

**assumes**  $eqExcPID-N s s1$  **and**  $pid \neq PID \vee PID \neq pid$   
**shows**  $getReviewersReviews s cid pid = getReviewersReviews s1 cid pid$   
**proof** –

**have**  
 $(\lambda uID. if isRev s cid uID pid then [(uID, getNthReview s pid (getReviewIndex s$   
 $cid uID pid))] else []) =$   
 $(\lambda uID. if isRev s1 cid uID pid then [(uID, getNthReview s1 pid (getReviewIndex$   
 $s1 cid uID pid))] else [])$   
**apply**  $(rule ext)$   
**using**  $assms$  **by**  $(auto simp: eqExcPID-N-imp eqExcPID-N-imp1)$   
**thus**  $?thesis$  **unfolding**  $getReviewersReviews-def$  **using**  $assms$  **by**  $(simp add:$   
 $eqExcPID-N-imp)$   
**qed**

**lemma**  $eqExcPID-N-imp3$ :

$eqExcPID-N s s1 \implies pid \neq PID \vee PID \neq pid \vee (n < length (reviewsPaper (paper$   
 $s PID)) \wedge n \neq N)$

$\implies$   
`getNthReview s pid n = getNthReview s1 pid n`  
**unfolding** `eqExcPID-N-def`  
**apply** `auto`  
**apply** (`metis eqExcPID-N-imp(2) getNthReview-def`)  
**unfolding** `eqExcPID-N-def` **apply** `simp` **unfolding** `eqExcD eqExcNth-def`  
**by** (`metis getNthReview-def`)

**lemma** `eqExcPID-N-imp3'`:  
**assumes** `s: reach s`  
**and** `eqExcPID-N s s1` **and** `pid  $\neq$  PID  $\vee$  (isRevNth s cid uid pid n  $\wedge$  n  $\neq$  N)`  
**shows** `getNthReview s pid n = getNthReview s1 pid n`  
**proof** –  
**have** `isRevNth s cid uid pid n  $\implies$  pid  $\neq$  PID  $\vee$  n < length (reviewsPaper (paper s PID))`  
**using** `s` **by** (`metis isRevNth-less-length`)  
**thus** `?thesis` **using** `eqExcPID-N-imp3' assms` **by** `auto`  
**qed**

**lemma** `eqExcPID-N-RDD`:  
`eqExcPID-N s s1  $\implies$`   
`titlePaper (paper s PID) = titlePaper (paper s1 PID)  $\wedge$`   
`abstractPaper (paper s PID) = abstractPaper (paper s1 PID)  $\wedge$`   
`contentPaper (paper s PID) = contentPaper (paper s1 PID)  $\wedge$`   
`disPaper (paper s PID) = disPaper (paper s1 PID)  $\wedge$`   
`decsPaper (paper s PID) = decsPaper (paper s1 PID)`  
**using** `eqExcPID-N-imp eqExcPID-N-RDD` **by** `auto`

**lemma** `eqExcPID-N-cong[simp, intro]`:  
 $\bigwedge uu1 uu2. eqExcPID-N s s1 \implies uu1 = uu2 \implies eqExcPID-N (s (\downarrow confIDs := uu1)) (s1 (\downarrow confIDs := uu2))$   
 $\bigwedge uu1 uu2. eqExcPID-N s s1 \implies uu1 = uu2 \implies eqExcPID-N (s (\downarrow conf := uu1)) (s1 (\downarrow conf := uu2))$

$\bigwedge uu1 uu2. eqExcPID-N s s1 \implies uu1 = uu2 \implies eqExcPID-N (s (\downarrow userIDs := uu1)) (s1 (\downarrow userIDs := uu2))$   
 $\bigwedge uu1 uu2. eqExcPID-N s s1 \implies uu1 = uu2 \implies eqExcPID-N (s (\downarrow pass := uu1)) (s1 (\downarrow pass := uu2))$   
 $\bigwedge uu1 uu2. eqExcPID-N s s1 \implies uu1 = uu2 \implies eqExcPID-N (s (\downarrow user := uu1)) (s1 (\downarrow user := uu2))$   
 $\bigwedge uu1 uu2. eqExcPID-N s s1 \implies uu1 = uu2 \implies eqExcPID-N (s (\downarrow roles := uu1)) (s1 (\downarrow roles := uu2))$

$\bigwedge uu1 uu2. eqExcPID-N s s1 \implies uu1 = uu2 \implies eqExcPID-N (s (\downarrow paperIDs := uu1)) (s1 (\downarrow paperIDs := uu2))$   
 $\bigwedge uu1 uu2. eqExcPID-N s s1 \implies eqExcPID-N uu1 uu2 \implies eqExcPID-N (s (\downarrow paper := uu1)) (s1 (\downarrow paper := uu2))$

$\wedge uu1 uu2. eqExcPID-N s s1 \implies uu1 = uu2 \implies eqExcPID-N (s (\backslash pref := uu1))$   
 $(s1 (\backslash pref := uu2))$   
 $\wedge uu1 uu2. eqExcPID-N s s1 \implies uu1 = uu2 \implies eqExcPID-N (s (\backslash voronkov := uu1))$   
 $(s1 (\backslash voronkov := uu2))$   
 $\wedge uu1 uu2. eqExcPID-N s s1 \implies uu1 = uu2 \implies eqExcPID-N (s (\backslash news := uu1))$   
 $(s1 (\backslash news := uu2))$   
 $\wedge uu1 uu2. eqExcPID-N s s1 \implies uu1 = uu2 \implies eqExcPID-N (s (\backslash phase := uu1))$   
 $(s1 (\backslash phase := uu2))$

**unfolding** *eqExcPID-N-def* **by** *auto*

**lemma** *eqExcPID-N-Paper*:  
**assumes** *s's1'*: *eqExcPID-N s s1*  
**and** *paper s pid = Paper title abstract content reviews dis decs*  
**and** *paper s1 pid = Paper title1 abstract1 content1 reviews1 dis1 decs1*  
**shows** *title = title1  $\wedge$  abstract = abstract1  $\wedge$  content = content1  $\wedge$  dis = dis1  $\wedge$  decs = decs1*  
**using** *assms unfolding eqExcPID-N-def apply (auto simp: eqExcD eqExcPID-N-def)*  
**by** (*metis titlePaper.simps abstractPaper.simps contentPaper.simps disPaper.simps decsPaper.simps*)+

Auxiliary definitions for a slightly weaker equivalence relation defined below:

**definition** *eqExcNth2* **where**  
*eqExcNth2 rl rl1 n  $\equiv$*   
*length rl = length rl1  $\wedge$*   
*( $\forall i < length rl. i \neq n \longrightarrow rl!i = rl1!i$ )  $\wedge$*   
*hd (rl!n) = hd (rl1!n)*

**lemma** *eqExcNth2-eq[simp,intro!]*: *eqExcNth2 rl rl n*  
**unfolding** *eqExcNth2-def* **by** *auto*

**lemma** *eqExcNth2-sym*:  
**assumes** *eqExcNth2 rl rl1 n*  
**shows** *eqExcNth2 rl1 rl n*  
**using** *assms unfolding eqExcNth2-def* **by** *auto*

**lemma** *eqExcNth2-trans*:  
**assumes** *eqExcNth2 rl rl1 n* **and** *eqExcNth2 rl1 rl2 n*  
**shows** *eqExcNth2 rl rl2 n*  
**using** *assms unfolding eqExcNth2-def* **by** *auto*

**fun** *eqExcD2* :: *paper  $\Rightarrow$  paper  $\Rightarrow$  bool **where**  
*eqExcD2 (Paper title abstract ct reviews dis decs)*  
*(Paper title1 abstract1 ct1 reviews1 dis1 decs1) =*  
*(title = title1  $\wedge$  abstract = abstract1  $\wedge$  ct = ct1  $\wedge$  dis = dis1  $\wedge$  decs = decs1  $\wedge$*   
*eqExcNth2 reviews reviews1 N)**

**lemma** *eqExcD2*:  
*eqExcD2 pap pap1 =*

$(titlePaper\ pap = titlePaper\ pap1 \wedge abstractPaper\ pap = abstractPaper\ pap1 \wedge$   
 $contentPaper\ pap = contentPaper\ pap1 \wedge$   
 $disPaper\ pap = disPaper\ pap1 \wedge decsPaper\ pap = decsPaper\ pap1 \wedge$   
 $eqExcNth2\ (reviewsPaper\ pap)\ (reviewsPaper\ pap1)\ N)$   
**by**(cases pap, cases pap1, auto)

**lemma** eqExcD2-eq[simp,intro!]: eqExcD2 pap pap  
**unfolding** eqExcD2 **using** eqExcNth2-eq **by** auto

**lemma** eqExcD2-sym:  
**assumes** eqExcD2 pap pap1  
**shows** eqExcD2 pap1 pap  
**using** assms **unfolding** eqExcD2 **using** eqExcNth2-sym **by** auto

**lemma** eqExcD2-trans:  
**assumes** eqExcD2 pap pap1 **and** eqExcD2 pap1 pap2  
**shows** eqExcD2 pap pap2  
**using** assms **unfolding** eqExcD2 **using** eqExcNth2-trans **by** auto

**definition** eeqExcPID-N2 **where**  
 $eeqExcPID-N2\ paps\ paps1 \equiv$   
 $\forall\ pid.\ if\ pid = PID\ then\ eqExcD2\ (paps\ pid)\ (paps1\ pid)\ else\ paps\ pid = paps1\ pid$

**lemma** eeqExcPID-N2-eeq[simp,intro!]: eeqExcPID-N2 s s  
**unfolding** eeqExcPID-N2-def **by** auto

**lemma** eeqExcPID-N2-sym:  
**assumes** eeqExcPID-N2 s s1 **shows** eeqExcPID-N2 s1 s  
**using** assms eqExcD2-sym **unfolding** eeqExcPID-N2-def **by** auto

**lemma** eeqExcPID-N2-trans:  
**assumes** eeqExcPID-N2 s s1 **and** eeqExcPID-N2 s1 s2 **shows** eeqExcPID-N2 s s2  
**using** assms eqExcD2-trans **unfolding** eeqExcPID-N2-def **by** simp blast

**lemma** eeqExcPID-N2-imp:  
 $eeqExcPID-N2\ paps\ paps1 \implies eqExcD2\ (paps\ PID)\ (paps1\ PID)$   
 $\llbracket eeqExcPID-N2\ paps\ paps1; pid \neq PID \rrbracket \implies paps\ pid = paps1\ pid$   
**unfolding** eeqExcPID-N2-def **by** auto

**lemma** eeqExcPID-N2-cong:  
**assumes** eeqExcPID-N2 paps paps1  
**and** pid = PID  $\implies eqExcD2\ uu\ uu1$   
**and** pid  $\neq$  PID  $\implies uu = uu1$   
**shows** eeqExcPID-N2 (paps (pid := uu)) (paps1 (pid := uu1))  
**using** assms **unfolding** eeqExcPID-N2-def **by** auto

**lemma** eeqExcPID-N2-RDD:  
 $eeqExcPID-N2\ paps\ paps1 \implies$

$titlePaper (paps\ PID) = titlePaper (paps1\ PID) \wedge$   
 $abstractPaper (paps\ PID) = abstractPaper (paps1\ PID) \wedge$   
 $contentPaper (paps\ PID) = contentPaper (paps1\ PID) \wedge$   
 $disPaper (paps\ PID) = disPaper (paps1\ PID) \wedge$   
 $decsPaper (paps\ PID) = decsPaper (paps1\ PID)$   
**using** *eeqExcPID-N2-def* **unfolding** *eqExcD2* **by** *auto*

A weaker state equivalence that allows differences in old versions of the score and comments of the review ( $N$ ,  $PID$ ). It is used for the confidentiality property that does not cover PC members in the discussion phase, when they will learn about scores and comments.

**definition** *eqExcPID-N2* :: *state*  $\Rightarrow$  *state*  $\Rightarrow$  *bool* **where**  
 $eqExcPID-N2\ s\ s1 \equiv$   
 $confIDs\ s = confIDs\ s1 \wedge conf\ s = conf\ s1 \wedge$   
 $userIDs\ s = userIDs\ s1 \wedge pass\ s = pass\ s1 \wedge user\ s = user\ s1 \wedge roles\ s = roles\ s1 \wedge$   
 $paperIDs\ s = paperIDs\ s1$   
 $\wedge$   
 $eeqExcPID-N2\ (paper\ s)\ (paper\ s1)$   
 $\wedge$   
 $pref\ s = pref\ s1 \wedge$   
 $voronkov\ s = voronkov\ s1 \wedge$   
 $news\ s = news\ s1 \wedge phase\ s = phase\ s1$

**lemma** *eqExcPID-N2-eq[simp,intro!]*: *eqExcPID-N2*  $s\ s$   
**unfolding** *eqExcPID-N2-def* **by** *auto*

**lemma** *eqExcPID-N2-sym*:  
**assumes** *eqExcPID-N2*  $s\ s1$  **shows** *eqExcPID-N2*  $s1\ s$   
**using** *assms eeqExcPID-N2-sym* **unfolding** *eqExcPID-N2-def* **by** *auto*

**lemma** *eqExcPID-N2-trans*:  
**assumes** *eqExcPID-N2*  $s\ s1$  **and** *eqExcPID-N2*  $s1\ s2$  **shows** *eqExcPID-N2*  $s\ s2$   
**using** *assms eeqExcPID-N2-trans* **unfolding** *eqExcPID-N2-def* **by** *auto*

Implications from *eqExcPID-N2*, including w.r.t. auxiliary operations:

**lemma** *eqExcPID-N2-imp*:  
 $eqExcPID-N2\ s\ s1 \implies$   
 $confIDs\ s = confIDs\ s1 \wedge conf\ s = conf\ s1 \wedge$   
 $userIDs\ s = userIDs\ s1 \wedge pass\ s = pass\ s1 \wedge user\ s = user\ s1 \wedge roles\ s = roles\ s1 \wedge$   
 $paperIDs\ s = paperIDs\ s1$   
 $\wedge$   
 $eeqExcPID-N2\ (paper\ s)\ (paper\ s1)$   
 $\wedge$   
 $pref\ s = pref\ s1 \wedge$   
 $voronkov\ s = voronkov\ s1 \wedge$   
 $news\ s = news\ s1 \wedge phase\ s = phase\ s1 \wedge$

$getAllPaperIDs\ s = getAllPaperIDs\ s1 \wedge$   
 $isRev\ s\ cid\ uid\ pid = isRev\ s1\ cid\ uid\ pid \wedge$   
 $getReviewIndex\ s\ cid\ uid\ pid = getReviewIndex\ s1\ cid\ uid\ pid \wedge$   
 $getRevRole\ s\ cid\ uid\ pid = getRevRole\ s1\ cid\ uid\ pid \wedge$   
 $length\ (reviewsPaper\ (paper\ s\ pid)) = length\ (reviewsPaper\ (paper\ s1\ pid))$   
**unfolding**  $eqExcPID-N2-def\ getAllPaperIDs-def$   
**unfolding**  $isRev-def\ getReviewIndex-def\ getRevRole-def$  **apply**  $auto$   
**unfolding**  $eeqExcPID-N2-def\ eqExcD2\ eqExcNth2-def$  **by**  $simp\ metis$

**lemma**  $eqExcPID-N2-imp1$ :  
 $eqExcPID-N2\ s\ s1 \implies eqExcD2\ (paper\ s\ pid)\ (paper\ s1\ pid)$   
 $eqExcPID-N2\ s\ s1 \implies pid \neq PID \vee PID \neq pid \implies$   
 $paper\ s\ pid = paper\ s1\ pid \wedge$   
 $getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$   
**unfolding**  $eqExcPID-N2-def\ getNthReview-def\ eeqExcPID-N2-def$   
**apply**  $auto$   
**by**  $(metis\ eqExcD2-eq)$

**lemma**  $eqExcPID-N2-imp2$ :  
**assumes**  $eqExcPID-N2\ s\ s1$  **and**  $pid \neq PID \vee PID \neq pid$   
**shows**  $getReviewersReviews\ s\ cid\ pid = getReviewersReviews\ s1\ cid\ pid$   
**proof** –  
**have**  
 $(\lambda uID. if\ isRev\ s\ cid\ uID\ pid\ then\ [(uID,\ getNthReview\ s\ pid\ (getReviewIndex\ s\ cid\ uID\ pid))] else\ []) =$   
 $(\lambda uID. if\ isRev\ s1\ cid\ uID\ pid\ then\ [(uID,\ getNthReview\ s1\ pid\ (getReviewIndex\ s1\ cid\ uID\ pid))] else\ [])$   
**apply**  $(rule\ ext)$   
**using**  $assms$  **by**  $(auto\ simp: eqExcPID-N2-imp\ eqExcPID-N2-imp1)$   
**thus**  $?thesis$  **unfolding**  $getReviewersReviews-def$  **using**  $assms$  **by**  $(simp\ add: eqExcPID-N2-imp)$   
**qed**

**lemma**  $eqExcPID-N2-eqExcPID-N$ :  
 $eqExcPID-N2\ s\ s1 \implies eqExcPID-N\ s\ s1$   
**unfolding**  $eqExcPID-N-def\ eqExcPID-N2-def\ eeqExcPID-N-def\ eeqExcPID-N2-def$   
 $eqExcD2\ eqExcD$   
**by**  $(auto\ simp: eqExcNth-def\ eqExcNth2-def)$

**lemma**  $eqExcPID-N2-imp3$ :  
 $eqExcPID-N2\ s\ s1 \implies pid \neq PID \vee PID \neq pid \vee (n < length\ (reviewsPaper\ (paper\ s\ PID)) \wedge n \neq N)$   
 $\implies$   
 $getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$   
**by**  $(metis\ eqExcPID-N2-eqExcPID-N\ eqExcPID-N-imp3)$

**lemma**  $eqExcPID-N2-imp3'$ :  
**assumes**  $s: reach\ s$   
**and**  $eqExcPID-N2\ s\ s1$  **and**  $pid \neq PID \vee (isRevNth\ s\ cid\ uid\ pid\ n \wedge n \neq N)$

**shows**  $getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$   
**by** (*metis* *assms* *eqExcPID-N2-eqExcPID-N* *eqExcPID-N-imp3'*)

**lemma** *eqExcPID-N2-imp33*:

**assumes** *eqExcPID-N2* *s* *s1*

**shows**  $hd\ (getNthReview\ s\ pid\ N) = hd\ (getNthReview\ s1\ pid\ N)$

**proof**(*cases* *pid = PID*)

**case** *False* **thus** ?*thesis* **using** *eqExcPID-N2-imp3*[*OF* *assms*] **by** *auto*

**next**

**case** *True* **thus** ?*thesis* **apply** *simp*

**using** *assms* **unfolding** *eqExcPID-N2-def* *eeqExcPID-N2-def* *eqExcD2* *eqExc-Nth2-def* *getNthReview-def* **by** *auto*

**qed**

**lemma** *eqExcPID-N2-RDD*:

*eqExcPID-N2* *s* *s1*  $\implies$

*titlePaper* (*paper* *s* *PID*) = *titlePaper* (*paper* *s1* *PID*)  $\wedge$

*abstractPaper* (*paper* *s* *PID*) = *abstractPaper* (*paper* *s1* *PID*)  $\wedge$

*contentPaper* (*paper* *s* *PID*) = *contentPaper* (*paper* *s1* *PID*)  $\wedge$

*disPaper* (*paper* *s* *PID*) = *disPaper* (*paper* *s1* *PID*)  $\wedge$

*decsPaper* (*paper* *s* *PID*) = *decsPaper* (*paper* *s1* *PID*)

**using** *eqExcPID-N2-imp* *eeqExcPID-N2-RDD* **by** *auto*

**lemma** *eqExcPID-N2-cong[simp, intro]*:

$\bigwedge uu1\ uu2. eqExcPID-N2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N2\ (s\ (\downarrow confIDs := uu1))\ (s1\ (\downarrow confIDs := uu2))$

$\bigwedge uu1\ uu2. eqExcPID-N2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N2\ (s\ (\downarrow conf := uu1))\ (s1\ (\downarrow conf := uu2))$

$\bigwedge uu1\ uu2. eqExcPID-N2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N2\ (s\ (\downarrow userIDs := uu1))\ (s1\ (\downarrow userIDs := uu2))$

$\bigwedge uu1\ uu2. eqExcPID-N2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N2\ (s\ (\downarrow pass := uu1))\ (s1\ (\downarrow pass := uu2))$

$\bigwedge uu1\ uu2. eqExcPID-N2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N2\ (s\ (\downarrow user := uu1))\ (s1\ (\downarrow user := uu2))$

$\bigwedge uu1\ uu2. eqExcPID-N2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N2\ (s\ (\downarrow roles := uu1))\ (s1\ (\downarrow roles := uu2))$

$\bigwedge uu1\ uu2. eqExcPID-N2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N2\ (s\ (\downarrow paperIDs := uu1))\ (s1\ (\downarrow paperIDs := uu2))$

$\bigwedge uu1\ uu2. eqExcPID-N2\ s\ s1 \implies eqExcPID-N2\ uu1\ uu2 \implies eqExcPID-N2\ (s\ (\downarrow paper := uu1))\ (s1\ (\downarrow paper := uu2))$

$\bigwedge uu1\ uu2. eqExcPID-N2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N2\ (s\ (\downarrow pref := uu1))\ (s1\ (\downarrow pref := uu2))$

$\bigwedge uu1\ uu2. eqExcPID-N2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N2\ (s\ (\downarrow voronkov := uu1))\ (s1\ (\downarrow voronkov := uu2))$

$\bigwedge uu1\ uu2. eqExcPID-N2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N2\ (s\ (\downarrow news := uu1))\ (s1\ (\downarrow news := uu2))$

$uu1$ ) (s1 (news := uu2))  
 $\wedge uu1 uu2. eqExcPID-N2 s s1 \implies uu1 = uu2 \implies eqExcPID-N2 (s (phase := uu1)) (s1 (phase := uu2))$

**unfolding**  $eqExcPID-N2-def$  **by** *auto*

**lemma**  $eqExcPID-N2-Paper$ :

**assumes**  $s's1'$ :  $eqExcPID-N2 s s1$

**and**  $paper s pid = Paper title abstract content reviews dis decs$

**and**  $paper s1 pid = Paper title1 abstract1 content1 reviews1 dis1 decs1$

**shows**  $title = title1 \wedge abstract = abstract1 \wedge content = content1 \wedge dis = dis1 \wedge decs = decs1$

**using**  $assms$  **unfolding**  $eqExcPID-N2-def$  **apply** (*auto simp: eqExcD2 eeqExcPID-N2-def*)

**by** (*metis titlePaper.simps abstractPaper.simps contentPaper.simps disPaper.simps decsPaper.simps*)<sup>+</sup>

**lemma**  $eqExcPID-N2-step$ :

**assumes**  $ss1$ :  $eqExcPID-N2 s s1$

**and**  $step$ :  $step s a = (ou, s')$

**and**  $step1$ :  $step s1 a = (ou1, s1')$

**and**  $s$ :  $reach s$  **and**  $r$ :  $isRevNth s cid uid PID N$

**shows**  $eqExcPID-N2 s' s1'$

**proof** –

**note**  $eqs = eqExcPID-N2-imp[OF ss1]$

**note**  $eqs' = eqExcPID-N2-imp1[OF ss1]$

**have**  $r$ :  $N < length (reviewsPaper (paper s PID))$  **using**  $s r$  **by** (*metis is-RevNth-less-length*)

**have**  $r1$ :  $N < length (reviewsPaper (paper s1 PID))$

**using**  $r eqs$  **unfolding**  $eeqExcPID-N2-def eqExcD2 eqExcNth2-def$  **by** *simp*

**note**  $simps[simp] = c-defs u-defs uu-defs r-defs l-defs Paper-dest-conv eqExcPID-N2-def eeqExcPID-N2-def eqExcD2 eqExcNth2-def$

**note**  $* = step step1 eqs eqs' r r1$

**then show**  $?thesis$

**proof** (*cases a*)

**case** (*Cact x1*)

**with**  $*$  **show**  $?thesis$

**proof** (*cases x1*)

**case** (*cReview x81 x82 x83 x84 x85*)

**with**  $Cact *$  **show**  $?thesis$

**by** (*clarsimp; metis (no-types, lifting) less-SucE nth-append-length right-cons-left*)

**qed** *auto*

**next**

**case** (*Uact x2*)

**with**  $*$  **show**  $?thesis$

**proof** (*cases x2*)

**case** (*uReview x71 x72 x73 x74 x75 x76*)

```

    with Uact * show ?thesis
      by (clarsimp; metis (no-types, lifting) nth-list-update nth-list-update-neq)
    qed auto
  next
  case (UUact x3)
  with * show ?thesis
  proof (cases x3)
    case (uuReview x31 x32 x33 x34 x35 x36)
    with UUact * show ?thesis
      by (clarsimp; smt (verit) list.sel(1) nth-list-update nth-list-update-neq)
    qed auto
  qed auto
qed

```

## 6.2 Value Setup

```

fun  $\varphi$  :: (state,act,out) trans  $\Rightarrow$  bool where
 $\varphi$  (Trans - (Uact (uReview cid uid p pid n rc)) ou -) =
  (pid = PID  $\wedge$  n = N  $\wedge$  ou = outOK)
|
 $\varphi$  (Trans - (UUact (uuReview cid uid p pid n rc)) ou -) =
  (pid = PID  $\wedge$  n = N  $\wedge$  ou = outOK)
|
 $\varphi$  - = False

```

```

lemma  $\varphi$ -def2:
 $\varphi$  (Trans s a ou s') =
  (ou = outOK  $\wedge$ 
  ( $\exists$  cid uid p rc.
    a = Uact (uReview cid uid p PID N rc)
     $\vee$ 
    a = UUact (uuReview cid uid p PID N rc)
  ))
apply(cases a)
subgoal by simp
subgoal for x2 apply (cases x2, auto) .
subgoal for x3 apply(cases x3, auto) .
by simp-all

```

```

lemma uReview-uuReview-step-eqExcPID-N:
assumes a:
a = Uact (uReview cid uid p PID N rc)  $\vee$ 
a = UUact (uuReview cid uid p PID N rc)
and step s a = (ou,s')
shows eqExcPID-N s s'
using assms unfolding eqExcPID-N-def eeqExcPID-N-def by (auto simp: u-defs
uu-defs eqExcNth-def)

```

```

lemma  $\varphi$ -step-eqExcPID-N:

```

```

assumes  $\varphi$ :  $\varphi$  (Trans  $s$   $a$   $ou$   $s'$ )
and  $s$ : step  $s$   $a$  = ( $ou$ ,  $s'$ )
shows eqExcPID-N  $s$   $s'$ 
using  $\varphi$  uReview-uuReview-step-eqExcPID-N[OF -  $s$ ] unfolding  $\varphi$ -def2 by blast

lemma eqExcPID-N-step:
assumes  $s's1'$ : eqExcPID-N  $s$   $s1$ 
and  $step$ : step  $s$   $a$  = ( $ou$ ,  $s'$ )
and  $step1$ : step  $s1$   $a$  = ( $ou1$ ,  $s1'$ )
shows eqExcPID-N  $s'$   $s1'$ 
proof -
  note  $eqs$  = eqExcPID-N-imp[OF  $s's1'$ ]
  note  $eqs'$  = eqExcPID-N-imp1[OF  $s's1'$ ]

  note  $simps[simp]$  = c-defs u-defs uu-defs r-defs l-defs Paper-dest-conv eqExcPID-N-def eqExcPID-N-def eqExcD eqExcNth-def
  note  $*$  = step step1 eqs eqs'

then show ?thesis
proof (cases a)
  case (Cact  $x1$ )
    with  $*$  show ?thesis
    proof (cases x1)
      case (cReview  $x81$   $x82$   $x83$   $x84$   $x85$ )
        with Cact  $*$  show ?thesis
        by (clarsimp; metis (no-types, lifting) less-SucE nth-append-length right-cons-left)
      qed auto
    next
      case (Uact  $x2$ )
        with  $*$  show ?thesis
        proof (cases x2)
          case (uReview  $x71$   $x72$   $x73$   $x74$   $x75$   $x76$ )
            with Uact  $*$  show ?thesis
            by (clarsimp; metis (no-types, lifting) nth-list-update nth-list-update-neq)
          qed auto
        next
          case (UUact  $x3$ )
            with  $*$  show ?thesis
            proof (cases x3)
              case (uuReview  $x31$   $x32$   $x33$   $x34$   $x35$   $x36$ )
                with UUact  $*$  show ?thesis
                by (clarsimp; metis (no-types, lifting) nth-list-update nth-list-update-neq)
              qed auto
            qed auto
          qed
        qed
      qed
    qed
  qed

lemma eqExcPID-N-step- $\varphi$ -imp:
assumes  $ss1$ : eqExcPID-N  $s$   $s1$ 
and  $step$ : step  $s$   $a$  = ( $ou$ ,  $s'$ ) and  $step1$ : step  $s1$   $a$  = ( $ou1$ ,  $s1'$ )

```

**and**  $\varphi$ :  $\varphi$  ( $\text{Trans } s \ a \ ou \ s'$ )  
**shows**  $\varphi$  ( $\text{Trans } s1 \ a \ ou1 \ s1'$ )  
**using** *assms unfolding*  $\varphi$ -def2 **by** (*auto simp add: u-defs uu-defs eqExcPID-N-imp*)

**lemma** *eqExcPID-N-step- $\varphi$* :  
**assumes**  $s's1'$ : *eqExcPID-N*  $s \ s1$   
**and** *step*:  $\text{step } s \ a = (ou, s')$  **and** *step1*:  $\text{step } s1 \ a = (ou1, s1')$   
**shows**  $\varphi$  ( $\text{Trans } s \ a \ ou \ s'$ ) =  $\varphi$  ( $\text{Trans } s1 \ a \ ou1 \ s1'$ )  
**by** (*metis eqExcPID-N-step- $\varphi$ -imp eqExcPID-N-sym assms*)

**lemma** *eqExcPID-N2-step- $\varphi$ -imp*:  
**assumes**  $ss1$ : *eqExcPID-N2*  $s \ s1$   
**and** *step*:  $\text{step } s \ a = (ou, s')$  **and** *step1*:  $\text{step } s1 \ a = (ou1, s1')$   
**and**  $r$ :  $N < \text{length}(\text{reviewsPaper}(\text{paper } s \ PID))$   
**and**  $\varphi$ :  $\varphi$  ( $\text{Trans } s \ a \ ou \ s'$ )  
**shows**  $\varphi$  ( $\text{Trans } s1 \ a \ ou1 \ s1'$ )  
**using** *assms unfolding*  $\varphi$ -def2 **by** (*auto simp add: u-defs uu-defs eqExcPID-N2-imp*)

**lemma** *eqExcPID-N2-step- $\varphi$* :  
**assumes**  $s$ : *reach*  $s$  **and**  $s1$ : *reach*  $s1$   
**and**  $ss1$ : *eqExcPID-N2*  $s \ s1$   
**and** *step*:  $\text{step } s \ a = (ou, s')$  **and** *step1*:  $\text{step } s1 \ a = (ou1, s1')$   
**shows**  $\varphi$  ( $\text{Trans } s \ a \ ou \ s'$ ) =  $\varphi$  ( $\text{Trans } s1 \ a \ ou1 \ s1'$ )  
**proof**(*cases*  $\exists \ cid \ uid. \text{isRevNth } s \ cid \ uid \ PID \ N$ )  
  **case** *False*  
  **hence**  $\neg \varphi$  ( $\text{Trans } s \ a \ ou \ s'$ ) **unfolding**  $\varphi$ -def2 **using** *step*  
  **by** (*auto simp add: u-defs uu-defs*) (*metis isRev-imp-isRevNth-getReviewIndex*)+  
  **moreover** **have**  $\neg \varphi$  ( $\text{Trans } s1 \ a \ ou1 \ s1'$ ) **using** *step1 False unfolding*  $\varphi$ -def2  
  **by** (*auto simp add: u-defs uu-defs*) (*metis eqExcPID-N2-def isRev-imp-isRevNth-getReviewIndex*  
   $ss1$ )+  
  **ultimately show** *?thesis* **by** *auto*  
**next**  
  **case** *True* **note**  $r = \text{True}$   
  **note**  $eqs = \text{eqExcPID-N2-imp}[OF \ ss1]$   
  **have**  $r$ :  $N < \text{length}(\text{reviewsPaper}(\text{paper } s \ PID))$   
  **using** *isRevNth-less-length[OF s] r* **by** *auto*  
  **have**  $r1$ :  $N < \text{length}(\text{reviewsPaper}(\text{paper } s1 \ PID))$   
  **using**  $eqs \ r$  **unfolding** *eeqExcPID-N2-def eqExcD2 eqExcNth2-def* **by** *simp*  
  **thus** *?thesis* **by** (*metis eqExcPID-N2-step- $\varphi$ -imp eqExcPID-N2-sym assms r*)  
**qed**

**end**  
**theory** *Review-RAut*  
**imports** *../Observation-Setup Review-Value-Setup Bounded-Deducibility-Security.Compositional-Reasoning*  
**begin**

### 6.3 Confidentiality protection from users who are not the review's author

We verify the following property:

A group UIDs of users learn nothing about the various updates of the N'th review of a paper PID except for the last edited version before discussion and all the later versions unless a user in UIDs is that review's author.

**type-synonym** *value* = *phase* \* *rcontent*

**fun** *f* :: (*state,act,out*) *trans* ⇒ *value* **where**  
*f* (*Trans s (Uact (uReview cid uid p pid n rc)) - -*) = (*phase s cid, rc*)  
|  
*f* (*Trans s (UUact (uuReview cid uid p pid n rc)) - -*) = (*phase s cid, rc*)

**fun** *T* :: (*state,act,out*) *trans* ⇒ *bool* **where**  
*T* (*Trans - - ou s'*) =  
(∃ *uid* ∈ *UIDs*. *isREVNth s' uid PID N*)

**declare** *T.simps* [*simp del*]

**definition** *B* :: *value list* ⇒ *value list* ⇒ *bool* **where**  
*B vl vl1* ≡  
∃ *ul ul1 wl*.  
*vl* = (*map (Pair revPH) ul*) @ (*map (Pair disPH) wl*) ∧  
*vl1* = (*map (Pair revPH) ul1*) @ (*map (Pair disPH) wl*) ∧  
*ul* ≠ [] ∧ *ul1* ≠ [] ∧ *last ul* = *last ul1*

**interpretation** *BD-Security-IO* **where**  
*istate* = *istate* **and** *step* = *step* **and**  
 $\varphi$  =  $\varphi$  **and** *f* = *f* **and**  $\gamma$  =  $\gamma$  **and** *g* = *g* **and** *T* = *T* **and** *B* = *B*  
**done**

**lemma** *reachNT-non-isRevNth*:  
**assumes** *reachNT s* **and** *uid* ∈ *UIDs*  
**shows** ¬ *isRevNth s cid uid PID N*  
**using** *assms*  
**apply** *induct*  
**apply** (*auto simp: istate-def*)[]  
**subgoal for** *trn* **apply**(*cases trn, auto simp: T.simps reachNT-reach isREVNth-def*)  
.  
**done**

**lemma** *P- $\varphi$ - $\gamma$* :  
**assumes** *1: reachNT s* **and** *2: step s a = (ou,s')  $\varphi$  (Trans s a ou s')*

**shows**  $\neg \gamma$  (*Trans s a ou s'*)  
**using** *reachNT-non-isRevNth[OF 1] 2 unfolding T.simps  $\varphi$ -def2*  
**apply** (*auto simp add: u-defs uu-defs*) **by** (*metis isRev-imp-isRevNth-getReviewIndex*)+

**lemma** *eqExcPID-N-step-out:*  
**assumes** *s's1': eqExcPID-N s s1*  
**and** *step: step s a = (ou,s')* **and** *step1: step s1 a = (ou1,s1')*  
**and** *sT: reachNT s* **and** *s1: reach s1*  
**and** *PID: PID  $\in\in$  paperIDs s cid*  
**and** *ph: phase s cid = revPH*  
**and** *UIDs: userOfA a  $\in$  UIDs*  
**shows** *ou = ou1*

**proof** –

**note** *Inv = reachNT-non-isRevNth[OF sT UIDs]*

**note** *eqs = eqExcPID-N-imp[OF s's1']*

**note** *eqs' = eqExcPID-N-imp1[OF s's1']*

**note** *s = reachNT-reach[OF sT]*

**note** *simps[simp] = c-defs u-defs uu-defs r-defs l-defs Paper-dest-conv eqExcPID-N-def eqExcPID-N-def eqExcD*

**note** *\* = step step1 eqs eqs' s s1 PID UIDs ph paperIDs-equals[OF s] Inv*

**show** *?thesis*

**proof** (*cases a*)

**case** (*Cact x1*)

**with** \* **show** *?thesis* **by** (*cases x1; auto*)

**next**

**case** (*Uact x2*)

**with** \* **show** *?thesis* **by** (*cases x2; auto*)

**next**

**case** (*UUact x3*)

**with** \* **show** *?thesis* **by** (*cases x3; auto*)

**next**

**case** (*Ract x4*)

**with** \* **show** *?thesis*

**proof** (*cases x4*)

**case** (*rMyReview x81 x82 x83 x84*)

**with** *Ract* \* **show** *?thesis*

**by** *clarsimp (metis eqExcPID-N-imp3' getRevRole-Some-Rev-isRevNth s's1')*

**next**

**case** (*rReviews x91 x92 x93 x94*)

**with** *Ract* \* **show** *?thesis*

**by** *clarsimp (metis Suc-n-not-le-n eqExcPID-N-imp2 s's1')*

**next**

**case** (*rFinalReviews x121 x122 x123 x124*)

**with** *Ract* \* **show** *?thesis*

**by** *clarsimp (metis Suc-leD Suc-n-not-le-n)*

**qed** *auto*

**next**

```

    case (Lact x5)
    with * show ?thesis by (cases x5; auto; presburger)
  qed
qed

```

```

lemma eqExcPID-N-imp-eq:
assumes eqExcPID-N paps paps1
and reviewsPaper (paps PID) ! N = reviewsPaper (paps1 PID) ! N
shows paps = paps1
proof(rule ext)
  fix pid
  show paps pid = paps1 pid
  using assms unfolding eqExcPID-N-def eqExcD eqExcNth-def
  apply(cases paps PID, cases paps1 PID, cases pid = PID)
  by simp-all (metis nth-equalityI)
qed

```

```

lemma eqExcPID-N-imp-eq:
assumes e: eqExcPID-N s s1
and reviewsPaper (paper s PID) ! N = reviewsPaper (paper s1 PID) ! N
shows s = s1
proof-
  have paper s = paper s1 using assms eqExcPID-N-imp-eq
  unfolding eqExcPID-N-def by metis
  thus ?thesis
  using e unfolding eqExcPID-N-def by (intro state.equality) auto
qed

```

```

lemma eqExcPID-N-step-eq:
assumes s: reach s and ss1: eqExcPID-N s s1
and a: a = Uact (uReview cid uid p PID N rc)
and step: step s a = (outOK, s') and step1: step s1 a = (ou', s1')
shows s' = s1'
proof(cases  $\exists$  cid uid. isRevNth s cid uid PID N)
  case False
  hence False
  using step unfolding a
  by (auto simp add: u-defs uu-defs) (metis isRev-imp-isRevNth-getReviewIndex)+
  thus ?thesis by auto
next
  case True note r = True
  note eqs = eqExcPID-N-imp[OF ss1]
  note eqsT = eqExcPID-N-Paper[OF ss1]
  have r: N < length (reviewsPaper (paper s PID))
  using isRevNth-less-length[OF s] r by auto
  have r1: N < length (reviewsPaper (paper s1 PID))
  using eqs r unfolding eqExcPID-N-def eqExcD eqExcNth-def by simp
  have s's1': eqExcPID-N s' s1' using assms by (metis eqExcPID-N-step)

```

**have**  $e\text{-updateReview } s \text{ cid uid } p \text{ PID } N \text{ rc}$   
**using**  $step \ a \ \mathbf{by} \ auto$   
**hence**  $e\text{-updateReview } s1 \text{ cid uid } p \text{ PID } N \text{ rc}$   
**using**  $eqExcPID\text{-}N\text{-imp}[OF \ ss1] \ u\text{-defs} \ \mathbf{by} \ auto$   
**hence**  $ou': ou' = outOK \ \mathbf{using} \ step1 \ a \ \mathbf{by} \ auto$

**let**  $?p = paper \ s \ \text{PID} \ \mathbf{let} \ ?p1 = paper \ s1 \ \text{PID}$

**have**  $1: eqExcD \ ?p \ ?p1$   
**using**  $ss1 \ eqExcPID\text{-}N\text{-imp} \ \mathbf{unfolding} \ eqExcPID\text{-}N\text{-def} \ \mathbf{by} \ auto$

**have**  $2: reviewsPaper \ (paper \ s' \ \text{PID}) \ ! \ N = reviewsPaper \ (paper \ s1' \ \text{PID}) \ ! \ N$   
**using**  $step \ step1[unfolded \ ou'] \ r \ r1 \ \mathbf{unfolding} \ a$   
**by**  $(cases \ ?p, \ cases \ ?p1) \ (auto \ simp : u\text{-defs})$

**from**  $1 \ 2 \ \mathbf{show} \ ?thesis \ \mathbf{using} \ eqExcPID\text{-}N\text{-imp}\text{-}eq \ s's1' \ \mathbf{by} \ blast$   
**qed**

**definition**  $\Delta1 :: state \Rightarrow value \ list \Rightarrow state \Rightarrow value \ list \Rightarrow bool \ \mathbf{where}$   
 $\Delta1 \ s \ vl \ s1 \ vl1 \equiv$   
 $(\forall \ cid. \ \text{PID} \in \in \ paperIDs \ s \ cid \longrightarrow \ phase \ s \ cid < revPH) \wedge$   
 $s = s1 \wedge B \ vl \ vl1$

**definition**  $\Delta2 :: state \Rightarrow value \ list \Rightarrow state \Rightarrow value \ list \Rightarrow bool \ \mathbf{where}$   
 $\Delta2 \ s \ vl \ s1 \ vl1 \equiv$   
 $\exists \ cid.$   
 $\text{PID} \in \in \ paperIDs \ s \ cid \wedge \ phase \ s \ cid = revPH \wedge \neg (\exists \ uid. \ isREVNth \ s \ uid$   
 $\text{PID } N) \wedge$   
 $s = s1 \wedge B \ vl \ vl1$

**definition**  $\Delta3 :: state \Rightarrow value \ list \Rightarrow state \Rightarrow value \ list \Rightarrow bool \ \mathbf{where}$   
 $\Delta3 \ s \ vl \ s1 \ vl1 \equiv$   
 $\exists \ cid \ uid.$   
 $\text{PID} \in \in \ paperIDs \ s \ cid \wedge \ phase \ s \ cid = revPH \wedge \ isREVNth \ s \ uid \ \text{PID } N \wedge$   
 $eqExcPID\text{-}N \ s \ s1 \wedge B \ vl \ vl1$

**definition**  $\Delta4 :: state \Rightarrow value \ list \Rightarrow state \Rightarrow value \ list \Rightarrow bool \ \mathbf{where}$   
 $\Delta4 \ s \ vl \ s1 \ vl1 \equiv$   
 $\exists \ cid \ uid.$   
 $\text{PID} \in \in \ paperIDs \ s \ cid \wedge \ phase \ s \ cid \geq revPH \wedge \ isREVNth \ s \ uid \ \text{PID } N \wedge$   
 $s = s1 \wedge (\exists \ wl. \ vl = map \ (Pair \ disPH) \ wl \wedge \ vl1 = map \ (Pair \ disPH) \ wl)$

**definition**  $\Delta e :: state \Rightarrow value \ list \Rightarrow state \Rightarrow value \ list \Rightarrow bool \ \mathbf{where}$   
 $\Delta e \ s \ vl \ s1 \ vl1 \equiv$   
 $vl \neq [] \wedge$   
 $($   
 $(\exists \ cid. \ \text{PID} \in \in \ paperIDs \ s \ cid \wedge \ phase \ s \ cid > revPH \wedge \neg (\exists \ uid. \ isREVNth \ s$   
 $uid \ \text{PID } N))$

$\vee$   
 $(\exists cid. PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > revPH \wedge fst\ (hd\ vl) = revPH)$   
 $)$

**lemma** *istate- $\Delta 1$* :  
**assumes**  $B: B\ vl\ vl1$   
**shows**  $\Delta 1\ istate\ vl\ istate\ vl1$   
**using**  $B$  **unfolding**  $\Delta 1$ -def  $B$ -def *istate-def* **by** *auto*

**lemma** *unwind-cont- $\Delta 1$* : *unwind-cont*  $\Delta 1\ \{\Delta 1, \Delta 2, \Delta e\}$   
**proof**(*rule, simp*)  
**let**  $?\Delta = \lambda s\ vl\ s1\ vl1. \Delta 1\ s\ vl\ s1\ vl1 \vee \Delta 2\ s\ vl\ s1\ vl1 \vee \Delta e\ s\ vl\ s1\ vl1$   
**fix**  $s\ s1 :: state$  **and**  $vl\ vl1 :: value\ list$   
**assume**  $rsT: reachNT\ s$  **and**  $rs1: reach\ s1$  **and**  $\Delta 1\ s\ vl\ s1\ vl1$   
**hence**  $rs: reach\ s$  **and**  $ss1: s1 = s$  **and**  $B: B\ vl\ vl1$   
**and**  $vl: vl \neq []$  **and**  $vl1: vl1 \neq []$  **and**  $PID$ -ph:  $\bigwedge cid. PID \in \in paperIDs\ s\ cid$   
 $\longrightarrow phase\ s\ cid < revPH$   
**using** *reachNT-reach* **unfolding**  $\Delta 1$ -def  $B$ -def **by** *auto*  
**show** *iaction*  $?\Delta\ s\ vl\ s1\ vl1 \vee$   
 $((vl = [] \longrightarrow vl1 = []) \wedge reaction\ ?\Delta\ s\ vl\ s1\ vl1)$  **(is**  $?iact \vee (- \wedge ?react)$ **)**  
**proof**-  
**have**  $?react$  **proof**  
**fix**  $a :: act$  **and**  $ou :: out$  **and**  $s' :: state$  **and**  $vl'$   
**let**  $?trn = Trans\ s\ a\ ou\ s'$  **let**  $?trn1 = Trans\ s1\ a\ ou\ s'$   
**assume** *step*:  $step\ s\ a = (ou, s')$  **and**  $T: \neg T\ ?trn$  **and**  $c: consume\ ?trn\ vl\ vl'$   
**have**  $\varphi: \neg \varphi\ ?trn$   
**apply**(*cases a*)  
**subgoal** **by** *simp*  
**subgoal** **for**  $x2$  **apply**(*cases x2*) **using** *step PID-ph* **by** (*fastforce simp:*  
*u-defs*)  
**subgoal** **for**  $x3$  **apply**(*cases x3*) **using** *step PID-ph* **by** (*fastforce simp:*  
*uu-defs*)  
**by** *simp-all*  
**hence**  $vl': vl' = vl$  **using**  $c$  **unfolding** *consume-def* **by** *auto*  
**show** *match*  $?\Delta\ s\ s1\ vl1\ a\ ou\ s'\ vl' \vee ignore\ ?\Delta\ s\ s1\ vl1\ a\ ou\ s'\ vl'$  **(is**  $?match$   
 $\vee ?ignore)$   
**proof**-  
**have**  $?match$  **proof**  
**show** *validTrans*  $?trn1$  **unfolding**  $ss1$  **using** *step* **by** *simp*  
**next**  
**show** *consume*  $?trn1\ vl1\ vl1$  **unfolding** *consume-def ss1* **using**  $\varphi$  **by** *auto*  
**next**  
**show**  $\gamma\ ?trn = \gamma\ ?trn1$  **unfolding**  $ss1$  **by** *simp*  
**next**  
**assume**  $\gamma\ ?trn$  **thus**  $g\ ?trn = g\ ?trn1$  **unfolding**  $ss1$  **by** *simp*  
**next**  
**show**  $?\Delta\ s'\ vl'\ s'\ vl1$   
**proof**(*cases*  $\exists cid. PID \in \in paperIDs\ s\ cid$ )  
**case** *False* **note**  $PID = False$

```

      have PID-ph':  $\bigwedge cid. PID \in \in paperIDs s' cid \implies phase s' cid < revPH$ 
using PID step rs
  apply(cases a)
    subgoal for - x1 apply(cases x1) apply(fastforce simp: c-defs)+ .
    subgoal for - x2 apply(cases x2) apply(fastforce simp: u-defs)+ .
    subgoal for - x3 apply(cases x3) apply(fastforce simp: uu-defs)+ .
    by auto
  hence  $\Delta 1 s' vl' s' vl1$  unfolding  $\Delta 1-def vl'$  using B PID-ph' vl by auto
  thus ?thesis by auto
next
case True
then obtain CID where PID:  $PID \in \in paperIDs s CID$  by auto
hence ph:  $phase s CID < revPH$  using PID-ph by auto
  have PID':  $PID \in \in paperIDs s' CID$  by (metis PID paperIDs-mono
step)
  show ?thesis
  proof(cases phase s' CID < revPH)
    case True note ph' = True
    hence  $\Delta 1 s' vl' s' vl1$  unfolding  $\Delta 1-def vl'$  using B vl ph' PID' apply
auto
    by (metis reach-PairI paperIDs-equals rs step)
    thus ?thesis by auto
  next
    case False note ph' = False
    have  $\neg (\exists uid. isRevNth s CID uid PID N)$  using rs ph is-
RevNth-geq-revPH by fastforce
    hence ph-isRev':  $phase s' CID = revPH \wedge \neg (\exists uid. isRevNth s' CID$ 
uid PID N)
    using ph' ph PID step rs
    apply(cases a)
      subgoal for x1 apply(cases x1) apply(fastforce simp: c-defs)+ .
      subgoal for x2 apply(cases x2) apply(fastforce simp: u-defs)+ .
      subgoal for x3 apply(cases x3) apply(fastforce simp: uu-defs)+ .
      by auto
    hence  $\neg (\exists uid. isREVNth s' uid PID N)$ 
    by (metis PID' isREVNth-imp-isRevNth reach-PairI rs step)
    hence  $\Delta 2 s' vl' s' vl1$  unfolding  $\Delta 2-def vl'$  using B vl ph' PID'
ph-isRev' by auto
    thus ?thesis by auto
  qed
qed
qed
qed
  thus ?thesis by simp
qed
qed
  thus ?thesis using vl by auto
qed
qed

```

**lemma** *unwind-cont- $\Delta 2$* : *unwind-cont*  $\Delta 2$   $\{\Delta 2, \Delta 3, \Delta e\}$   
**proof**(*rule, simp*)  
**let**  $?\Delta = \lambda s \text{ vl } s1 \text{ vl1}. \Delta 2 \text{ s vl s1 vl1} \vee \Delta 3 \text{ s vl s1 vl1} \vee \Delta e \text{ s vl s1 vl1}$   
**fix**  $s \text{ s1} :: \text{state}$  **and**  $\text{vl } \text{vl1} :: \text{value list}$   
**assume**  $rsT$ : *reachNT*  $s$  **and**  $rs1$ : *reach*  $s1$  **and**  $\Delta 2 \text{ s vl s1 vl1}$   
**then obtain**  $CID$  **where**  $uid$ :  $\neg (\exists \text{ uid}. \text{isREVNth } s \text{ uid } PID \text{ } N)$  **and**  $PID$ :  
 $PID \in \in \text{paperIDs } s \text{ } CID$   
**and**  $rs$ : *reach*  $s$  **and**  $ph$ : *phase*  $s \text{ } CID = \text{revPH (is ?ph = -)}$  **and**  $ss1$ :  $s1 = s$   
**and**  $B$ :  $B \text{ vl } \text{vl1}$   
**and**  $\text{vl}$ :  $\text{vl} \neq []$  **and**  $\text{vl1}$ :  $\text{vl1} \neq []$   
**using** *reachNT-reach unfolding*  $\Delta 2\text{-def } B\text{-def}$  **by** *auto*  
**hence**  $uid$ :  $\neg (\exists \text{ uid}. \text{isRevNth } s \text{ } CID \text{ uid } PID \text{ } N)$  **by** (*metis isREVNth-def*)  
**show** *iaction*  $?\Delta \text{ s vl s1 vl1} \vee$   
 $((\text{vl} = [] \longrightarrow \text{vl1} = [])) \wedge \text{reaction } ?\Delta \text{ s vl s1 vl1}$  (**is**  $?iact \vee (- \wedge ?react)$ )  
**proof**–  
**have**  $?react$  **proof**  
**fix**  $a :: \text{act}$  **and**  $ou :: \text{out}$  **and**  $s' :: \text{state}$  **and**  $\text{vl}'$   
**let**  $?trn = \text{Trans } s \text{ } a \text{ } ou \text{ } s'$  **let**  $?trn1 = \text{Trans } s1 \text{ } a \text{ } ou \text{ } s'$   
**let**  $?ph' = \text{phase } s' \text{ } CID$   
**assume**  $step$ : *step*  $s \text{ } a = (ou, s')$  **and**  $T$ :  $\neg T \text{ } ?trn$  **and**  $c$ : *consume*  $?trn \text{ vl } \text{vl}'$   
**have**  $\varphi$ :  $\neg \varphi \text{ } ?trn$   
**apply**(*cases*  $a$ )  
**subgoal by** *simp*  
**subgoal for**  $x2$  **apply**(*cases*  $x2$ )  
**using**  $step \text{ } ph$  **apply** (*auto simp: u-defs*)  
**by** (*metis PID isRev-imp-isRevNth-getReviewIndex paperIDs-equals rs uid*)  
**subgoal for**  $x3$  **apply**(*cases*  $x3$ )  
**using**  $step \text{ } ph$  **apply** (*auto simp: uu-defs*)  
**using**  $PID \text{ paperIDs-equals } rs$  **by** *force*  
**by** *simp-all*  
**hence**  $\text{vl}'$ :  $\text{vl}' = \text{vl}$  **using**  $c$  **unfolding** *consume-def* **by** *auto*  
**have**  $PID'$ :  $PID \in \in \text{paperIDs } s' \text{ } CID$  **by** (*metis paperIDs-mono step PID*)  
**show** *match*  $?\Delta \text{ s s1 vl1 } a \text{ } ou \text{ } s' \text{ vl}' \vee \text{ignore } ?\Delta \text{ s s1 vl1 } a \text{ } ou \text{ } s' \text{ vl}'$  (**is**  $?match$   
 $\vee ?ignore$ )  
**proof**–  
**have**  $?match$  **proof**  
**show** *validTrans*  $?trn1$  **unfolding**  $ss1$  **using**  $step$  **by** *simp*  
**next**  
**show** *consume*  $?trn1 \text{ vl1 } \text{vl1}$  **unfolding** *consume-def*  $ss1$  **using**  $\varphi$  **by** *auto*  
**next**  
**show**  $\gamma \text{ } ?trn = \gamma \text{ } ?trn1$  **unfolding**  $ss1$  **by** *simp*  
**next**  
**assume**  $\gamma \text{ } ?trn$  **thus**  $g \text{ } ?trn = g \text{ } ?trn1$  **unfolding**  $ss1$  **by** *simp*  
**next**  
**show**  $?\Delta \text{ s' vl}' \text{ s' vl1}$   
**proof**(*cases*  $?ph' = \text{revPH}$ )  
**case** *True* **note**  $ph' = \text{True}$   
**show**  $?thesis$  **proof**(*cases*  $\exists \text{ uid}. \text{isRevNth } s' \text{ } CID \text{ uid } PID \text{ } N$ )  
**case** *False*

hence  $\neg (\exists \text{ uid. isREVNth } s' \text{ uid } PID \ N)$   
 by (metis *PID' isREVNth-def isRevNth-paperIDs paperIDs-equals reach-PairI rs1 ss1 step*)  
 hence  $\Delta 2 \ s' \ vl' \ s' \ vl1$  **unfolding**  $\Delta 2\text{-def } vl'$  **using** *B ph' PID' unfolding B-def* **by** *auto*  
 thus *?thesis* **by** *auto*  
 next  
 case *True* **hence**  $\exists \text{ uid. isREVNth } s' \text{ uid } PID \ N$  **by** (metis *isREVNth-def*)  
 hence  $\Delta 3 \ s' \ vl' \ s' \ vl1$  **unfolding**  $\Delta 3\text{-def } vl'$  **using** *B ph' PID' unfolding B-def* **by** *auto*  
 thus *?thesis* **by** *auto*  
 qed  
 next  
 case *False*  
 hence *ph'*: *?ph' > revPH* **by** (metis *le-less step ph phase-increases*)  
 hence  $\neg (\exists \text{ uid. isRevNth } s' \text{ CID } \text{uid } PID \ N)$  **using** *PID uid ph step rs apply(cases a)*  
 subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp: c-defs*)**+** .  
 subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp: u-defs*)**+** .  
 subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp: uu-defs*)**+** .  
 by *auto*  
 hence  $\neg (\exists \text{ uid. isREVNth } s' \text{ uid } PID \ N)$   
 by (metis *IO-Automaton.reach-PairI PID' isREVNth-imp-isRevNth rs1 ss1 step*)  
 hence  $\Delta e \ s' \ vl' \ s' \ vl1$  **using** *ph' vl PID' unfolding  $\Delta e\text{-def } vl'$*  **by** *auto*  
 thus *?thesis* **by** *auto*  
 qed  
 qed  
 thus *?thesis* **by** *simp*  
 qed  
 qed  
 thus *?thesis using vl* **by** *auto*  
 qed  
 qed

**lemma** *unwind-cont- $\Delta 3$ : unwind-cont  $\Delta 3 \ \{\Delta 3, \Delta 4, \Delta e\}$*

**proof**(*rule, simp*)

**let**  $?\Delta = \lambda s \ vl \ s1 \ vl1. \Delta 3 \ s \ vl \ s1 \ vl1 \ \vee \ \Delta 4 \ s \ vl \ s1 \ vl1 \ \vee \ \Delta e \ s \ vl \ s1 \ vl1$

**fix** *s s1 :: state* **and** *vl vl1 :: value list*

**assume** *rsT: reachNT s* **and** *rs1: reach s1* **and**  $\Delta 3 \ s \ vl \ s1 \ vl1$

**then obtain** *CID uid ul ul1 wl* **where** *uuid: isREVNth s uid PID N*

**and** *rs: reach s* **and** *ph: phase s CID = revPH (is ?ph = -)* **and** *ss1: eqExcPID-N s s1*

**and** *PID: PID  $\in \in$  paperIDs s CID* **and** *B: B vl vl1*

**and** *vlvl1: vl  $\neq$  [] vl1  $\neq$  []*

**and** *vl-wl: vl = (map (Pair revPH) ul) @ (map (Pair disPH) wl)*

**and** *vl1-wl: vl1 = (map (Pair revPH) ul1) @ (map (Pair disPH) wl)*

**and** *ulul1: ul  $\neq$  []  $\wedge$  ul1  $\neq$  []  $\wedge$  last ul = last ul1*

**using** *reachNT-reach unfolding  $\Delta 3\text{-def } B\text{-def}$*  **by** *blast*

**hence**  $uid: isRevNth\ s\ CID\ uid\ PID\ N$  **by** (*metis isREVNth-imp-isRevNth*)  
**have**  $ph1: phase\ s1\ CID = revPH$  **using**  $ss1\ ph\ eqExcPID-N-imp$  **by** *auto*  
**from**  $ulul1$  **obtain**  $u\ ul'\ u1\ ul1'$  **where**  $ul: ul = u \# ul'$  **and**  $ul1: ul1 = u1 \# ul1'$  **by** (*metis list.exhaust*)  
**obtain**  $vl'\ vl1'$  **where**  
 $vl: vl = (revPH, u) \# vl'$  **and**  $vl'-wl: vl' = (map\ (Pair\ revPH)\ ul') \ @\ (map\ (Pair\ disPH)\ wl)$  **and**  
 $vl1: vl1 = (revPH, u1) \# vl1'$  **and**  $vl1'-wl: vl1' = (map\ (Pair\ revPH)\ ul1') \ @\ (map\ (Pair\ disPH)\ wl)$   
**unfolding**  $vl-wl\ ul\ vl1-wl\ ul1$  **by** *auto*  
**have**  $uid-notin: uid \notin UIDs$  **using**  $uid$  **by** (*metis reachNT-non-isRevNth rsT*)  
**show**  $iact\ ?\Delta\ s\ vl\ s1\ vl1 \vee ((vl = [] \longrightarrow vl1 = []) \wedge reaction\ ?\Delta\ s\ vl\ s1\ vl1)$  (**is**  $?iact \vee (- \wedge ?react)$ )  
**proof**(*cases*  $ul1' = []$ )  
**case** *False* **note**  $ul1' = False$   
**hence**  $ul-ul1'$ :  $last\ ul = last\ ul1'$  **using**  $ulul1$  **unfolding**  $ul1$  **by** *simp*  
**have**  $uid1: isRevNth\ s1\ CID\ uid\ PID\ N$  **using**  $ss1\ uid$  **unfolding**  $eqExcPID-N-def$  **by** *auto*  
**define**  $a1$  **where**  $a1 \equiv Uact\ (uReview\ CID\ uid\ (pass\ s\ uid)\ PID\ N\ u1)$   
**obtain**  $s1'\ ou1$  **where**  $step1: step\ s1\ a1 = (ou1, s1')$  **by** (*metis prod.exhaust*)  
**let**  $?trn1 = Trans\ s1\ a1\ ou1\ s1'$   
**have**  $s1s1'$ :  $eqExcPID-N\ s1\ s1'$  **using**  $a1-def\ step1$  **by** (*metis uReview-uuReview-step-eqExcPID-N*)  
**have**  $ss1'$ :  $eqExcPID-N\ s\ s1'$  **using**  $eqExcPID-N-trans[OF\ ss1\ s1s1']$ .  
**hence**  $many-s1'$ :  $PID \in \in paperIDs\ s1'\ CID\ isRevNth\ s1'\ CID\ uid\ PID\ N$   
 $phase\ s1'\ CID = revPH\ pass\ s1'\ uid = pass\ s\ uid$   
**using**  $uid\ PID\ ph$  **unfolding**  $eqExcPID-N-def$  **by** *auto*  
**hence**  $more-s1'$ :  $uid \in \in userIDs\ s1'\ CID \in \in confIDs\ s1'$   
**by** (*metis paperIDs-confIDs reach-PairI roles-userIDs rs1 step1 many-s1'(1)*)  
**have**  $f: f\ ?trn1 = (revPH, u1)$  **unfolding**  $a1-def$  **using**  $ph1$  **by** *simp*  
**have**  $rs1'$ :  $reach\ s1'$  **using**  $rs1\ step1$  **by** (*auto intro: reach-PairI*)  
**have**  $ou1: ou1 = outOK$   
**using**  $step1\ uid1\ ph$  **unfolding**  $a1-def$  **apply** (*auto simp add: u-defs many-s1' more-s1'*)  
**by** (*metis isRevNth-getReviewIndex isRev-def3 many-s1'(2) rs1'*)  
**have**  $?iact$  **proof**  
**show**  $step\ s1\ a1 = (ou1, s1')$  **by** *fact*  
**next**  
**show**  $\varphi: \varphi\ ?trn1$  **using**  $ou1$  **unfolding**  $a1-def$  **by** *simp*  
**thus**  $consume\ ?trn1\ vl1\ vl1'$  **using**  $f$  **unfolding**  $consume-def\ vl1\ ul1$  **by** *simp*  
**next**  
**show**  $\neg\ \gamma\ ?trn1$  **by** (*simp add: a1-def uid-notin*)  
**next**  
**have**  $\Delta3\ s\ vl\ s1'\ vl1'$  **unfolding**  $\Delta3-def\ B-def$   
**using**  $ph\ PID\ ss1'\ uvid\ ul1'\ vl-wl\ vl1'-wl\ ulul1\ ul-ul1'$  **by** *fastforce*  
**thus**  $?\Delta\ s\ vl\ s1'\ vl1'$  **by** *simp*  
**qed**  
**thus**  $?thesis$  **by** *auto*  
**next**  
**case** *True* **hence**  $ul1: ul1 = [u1]$  **unfolding**  $ul1$  **by** *simp*

```

have ?react proof
  fix a :: act and ou :: out and s' :: state and vll'
  let ?trn = Trans s a ou s'
  let ?ph' = phase s' CID
  assume step: step s a = (ou, s') and T:  $\neg T$  ?trn and c: consume ?trn vl
vll'
  have PID': PID  $\in$  paperIDs s' CID using PID rs by (metis paperIDs-mono
step)
  have uid': isRevNth s' CID uid PID N by (metis isRevNth-persistent rs step
uid)
  hence uuid': isREVNth s' uid PID N by (metis isREVNth-def)
  show match ? $\Delta$  s s1 vl1 a ou s' vll'  $\vee$  ignore ? $\Delta$  s s1 vl1 a ou s' vll' (is
?match  $\vee$  ?ignore)
  proof(cases  $\varphi$  ?trn)
    case False note  $\varphi = \text{False}$ 
    have vll': vll' = vl using c  $\varphi$  unfolding consume-def by auto
    obtain ou1 and s1' where step1: step s1 a = (ou1, s1') by (metis
prod.exhaust)
    let ?trn1 = Trans s1 a ou1 s1'
    have s's1': eqExcPID-N s' s1' using eqExcPID-N-step[OF ss1 step step1] .
    have  $\varphi$ 1:  $\neg \varphi$  ?trn1 using  $\varphi$  unfolding eqExcPID-N-step- $\varphi$ [OF ss1 step
step1] .
    have ?match proof
      show validTrans ?trn1 using step1 by simp
    next
      show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi$ 1 by auto
    next
      show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
    next
      assume  $\gamma$  ?trn thus g ?trn = g ?trn1
      using eqExcPID-N-step-out[OF ss1 step step1 rsT rs1 PID ph] by simp
    next
      show ? $\Delta$  s' vll' s1' vl1
      proof(cases ?ph' = revPH)
        case True
          hence  $\Delta$ 3 s' vll' s1' vl1
          unfolding  $\Delta$ 3-def B-def vll' using ph PID s's1' PID' uuid' vl-wl vl1-wl
ulul1 by fastforce
          thus ?thesis by auto
        next
          case False hence ?ph' > revPH using ph rs step by (metis le-less
phase-increases2 snd-conv)
          hence  $\Delta$ e s' vll' s1' vl1 using vlvl1 PID' unfolding  $\Delta$ e-def vll' vl-wl ul
by auto
          thus ?thesis by auto
        qed
      qed
      thus ?thesis by simp
    next

```

```

case True note  $\varphi = \text{True}$ 
hence  $vll'$ :  $vll' = vl'$  using c unfolding vl consume-def by simp
obtain cid uid' p rc
where  $a = \text{Uact } (u\text{Review } cid \ uid' \ p \ PID \ N \ rc) \vee$ 
 $a = \text{UUact } (uu\text{Review } cid \ uid' \ p \ PID \ N \ rc)$  and  $ou$ :  $ou = \text{outOK}$ 
using  $\varphi \ c \ ph \ step \ ph$  unfolding vl consume-def  $\varphi\text{-def2}$   $vll'$  by force
hence  $a$ :  $a = \text{Uact } (u\text{Review } cid \ uid' \ p \ PID \ N \ rc)$ 
using step ph unfolding  $ou$  apply (auto simp: uu-defs) using PID paperIDs-equals rs by force
have cid:  $cid = CID$ 
using step unfolding  $a$ 
apply (simp add: u-defs uu-defs)
apply (metis PID e-updateReview-def isRev-paperIDs paperIDs-equals rs)
by (metis ou out.distinct(1))
hence  $\gamma$ :  $\neg \gamma \ ?trn$  using step T rsT by (metis P- $\varphi$ - $\gamma$  True)
hence  $f$ :  $f \ ?trn = (revPH, u)$  using c  $\varphi$  ph unfolding consume-def vl by
simp
have  $u$ :  $u = rc$  using  $f$  unfolding  $a$  by (auto simp: u-defs)
have  $s's$ : eqExcPID-N  $s' \ s$  using eqExcPID-N-sym[OF  $\varphi\text{-step-eqExcPID-N}$ [OF
 $\varphi \ step$ ]].
have  $s's1$ : eqExcPID-N  $s' \ s1$  using eqExcPID-N-trans[OF  $s's \ ss1$ ].
have  $ph'$ : phase  $s' \ CID = revPH$  using  $s's \ ph$  unfolding eqExcPID-N-def
by auto
show ?thesis
proof(cases  $ul' = []$ )
case False note  $ul' = \text{False}$ 
hence  $ul'ul1$ : last  $ul' = \text{last } ul1$  using ulul1 unfolding  $ul$  by auto
have ?ignore proof
show  $\neg \gamma \ ?trn$  by fact
next
show  $\Delta \ s' \ vll' \ s1 \ vl1$ 
proof(cases  $?ph' = revPH$ )
case True
hence  $\Delta \ s' \ vll' \ s1 \ vl1$ 
unfolding  $\Delta \ s\text{-def}$   $B\text{-def}$  using ph PID  $s's1 \ PID'$ 
apply – apply(rule exI[of – CID]) apply(rule exI[of – uid])
apply safe
subgoal using uuid' by simp
subgoal
apply(rule exI[of –  $ul'$ ]) apply(rule exI[of –  $ul1$ ]) apply(rule exI[of
–  $wl$ ])
unfolding  $vll'$  using  $vl'\text{-}wl \ vl1\text{-}wl \ ul'ul1 \ ul' \ ulul1$  by auto
done
thus ?thesis by auto
next
case False hence  $?ph' > revPH$  using rs step  $ph'$  by blast
hence  $\Delta \ e \ s' \ vll' \ s1 \ vl1$  unfolding  $\Delta \ e\text{-def}$   $vll' \ vl'\text{-}wl$  using  $ul' \ PID'$ 
by (cases  $ul'$ ) auto
thus ?thesis by auto

```

```

    qed
  qed
  thus ?thesis by auto
next
  case True note ul' = True hence ul: ul = [u] unfolding ul by simp

  hence u1u: u1 = u using ulul1 unfolding ul1 by simp
obtain s1' ou1 where step1: step s1 a = (ou1, s1') by (metis prod.exhaust)
  let ?trn1 = Trans s1 a ou1 s1'
  have  $\varphi 1$ :  $\varphi$  ?trn1 using eqExcPID-N-step- $\varphi$ -imp[OF ss1 step step1  $\varphi$ ] .
  hence ou1: ou1 = outOK unfolding  $\varphi$ -def2 by auto
  have PID  $\in\in$  paperIDs s1 CID  $\exists$  uid. isRevNth s1 CID uid PID N
  using eqExcPID-N-imp[OF ss1] PID uid by auto
  hence many-s1': revPH  $\leq$  phase s1' CID PID  $\in\in$  paperIDs s1' CID
   $\exists$  uid. isRevNth s1' CID uid PID N
  by (metis ph1 phase-increases step1 paperIDs-mono a
      eqExcPID-N-step-eq ou rs ss1 step step1 uid')+
  hence uuid1':  $\exists$  uid. isREVNth s1' uid PID N by (metis isREVNth-def)
  have ?match proof
  show validTrans ?trn1 using step1 by simp
next
  show consume ?trn1 vl1 (map (Pair disPH) wl)
  using  $\varphi 1$  ph1 unfolding consume-def by (simp add: a ul1 vl1-wl ul1 u1u
u ph1 cid)
next
  show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
next
  assume  $\gamma$  ?trn thus g ?trn = g ?trn1
  using eqExcPID-N-step-out[OF ss1 step step1 rsT rs1 PID ph] by simp
next
  note s's1' = eqExcPID-N-step-eq[OF rs ss1 a step[unfolded ou] step1]
  have  $\Delta 4$  s' vll' s1' (map (Pair disPH) wl)
  unfolding  $\Delta 4$ -def B-def using ph PID s's1' many-s1' uuid1'
  unfolding vll' vl'-wl ul' by auto
  thus ? $\Delta$  s' vll' s1' (map (Pair disPH) wl) by simp
  qed
  thus ?thesis by simp
  qed
  qed
  qed
  thus ?thesis using vl by auto
  qed
  qed

```

lemma unwind-cont- $\Delta 4$ : unwind-cont  $\Delta 4$  { $\Delta 4, \Delta e$ }

proof(rule, simp)

let ? $\Delta$  =  $\lambda s vl s1 vl1. \Delta 4 s vl s1 vl1 \vee \Delta e s vl s1 vl1$

fix s s1 :: state and vl vl1 :: value list

assume rsT: reachNT s and rs1: reach s1 and  $\Delta 4 s vl s1 vl1$

```

then obtain uid CID wl where uuid: isREVNth s uid PID N
and rs: reach s and ph: phase s CID ≥ revPH (is ?ph ≥ revPH) and ss1: s1
= s
and PID: PID ∈∈ paperIDs s CID and vl: vl = map (Pair disPH) wl
and vl1: vl1 = map (Pair disPH) wl
using reachNT-reach unfolding Δ4-def by blast
hence uid: isRevNth s CID uid PID N by (metis isREVNth-imp-isRevNth)
show iaction ?Δ s vl s1 vl1 ∨
  ((vl = [] → vl1 = []) ∧ reaction ?Δ s vl s1 vl1) (is ?iact ∨ (- ∧ ?react))
proof-
  have ?react
  proof
    fix a :: act and ou :: out and s' :: state and vl'
    let ?trn = Trans s a ou s'
    let ?ph' = phase s' CID
    assume step: step s a = (ou, s') and T: ¬ T ?trn and c: consume ?trn vl vl'
    have uid': isRevNth s' CID uid PID N by (metis isRevNth-persistent rs step
uid)
    hence uuid': isREVNth s' uid PID N by (metis isREVNth-def)
    have PID': PID ∈∈ paperIDs s' CID using PID rs by (metis paperIDs-mono
step)
    have ph': phase s' CID ≥ revPH using rs isRevNth-geq-revPH local.step
reach-PairI uid' by blast
    let ?trn1 = Trans s1 a ou s'
    show match ?Δ s s1 vl1 a ou s' vl' ∨ ignore ?Δ s s1 vl1 a ou s' vl' (is ?match
∨ ?ignore)
    proof(cases φ ?trn)
    case True note φ = True
    hence φ1: φ ?trn1 unfolding ss1 by simp
    obtain w wl' where wl: wl = w # wl' and vl: vl = (disPH,w) # map
(Pair disPH) wl'
    and vl1: vl1 = (disPH,w) # map (Pair disPH) wl' and vl': vl' = map
(Pair disPH) wl'
    using φ φ1 c unfolding vl vl1 consume-def by (cases wl) auto
    have f: f ?trn = (disPH, w) f ?trn1 = (disPH, w)
    using φ φ1 c unfolding consume-def vl vl1 ss1 by auto
    have ?match proof
      show validTrans ?trn1 using step unfolding ss1 by simp
    next
    show consume ?trn1 vl1 vl' unfolding consume-def vl1 vl' using φ1 f by
auto
  next
    show γ ?trn = γ ?trn1 unfolding ss1 by simp
  next
    assume γ ?trn thus g ?trn = g ?trn1 by simp
  next
    have Δ4 s' vl' s' vl'
    using ph' PID' uuid' unfolding Δ4-def vl' by auto
    thus ?Δ s' vl' s' vl' by simp

```

```

qed
thus ?thesis by simp
next
case False note  $\varphi = \text{False}$ 
hence  $\varphi 1: \neg \varphi$  ?trn1 unfolding ss1 by simp
hence  $vl': vl' = vl$  using  $\varphi c$  unfolding vl consume-def by auto
have ?match proof
  show validTrans ?trn1 using step unfolding ss1 by simp
next
  show consume ?trn1 vl1 vl unfolding consume-def vl1 vl using  $\varphi 1$  by
auto
next
  show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
next
  assume  $\gamma$  ?trn thus  $g$  ?trn =  $g$  ?trn1 by simp
next
  have  $\Delta_4 s' vl' s' vl$ 
  using  $ph' PID' uuid'$  unfolding  $\Delta_4$ -def vl' vl by auto
  thus  $?\Delta s' vl' s' vl$  by simp
qed
thus ?thesis by simp
qed
qed
thus ?thesis using vl vl1 by simp
qed
qed

```

**definition** *K2exit* where

*K2exit* cid s  $\equiv$

$PID \in \text{paperIDs } s \text{ cid} \wedge \text{phase } s \text{ cid} > \text{revPH} \wedge \neg (\exists \text{uid. isRevNth } s \text{ cid uid } PID \ N)$

**lemma** *invarNT-K2exit*: *invarNT* (*K2exit* cid)

**unfolding** *invarNT-def* **apply** (*safe dest!*: *reachNT-reach*)

**subgoal for - a** **apply**(*cases a*)

**subgoal for x1** **apply**(*cases x1*) **apply** (*fastforce simp add: c-defs K2exit-def geq-noPH-confIDs*) $+$  .

**subgoal for x2** **apply**(*cases x2*) **apply** (*fastforce simp add: u-defs K2exit-def paperIDs-equals*) $+$  .

**subgoal for x3** **apply**(*cases x3*) **apply** (*fastforce simp add: uu-defs K2exit-def*) $+$

.

**by** *auto*

**done**

**lemma** *noVal-K2exit*: *noVal* (*K2exit* cid) v

**apply**(*rule no $\varphi$ -noVal*)

**unfolding** *no $\varphi$ -def* **apply** *safe*

**subgoal for - a** **apply**(*cases a*)

**subgoal by** (*fastforce simp add: c-defs K2exit-def*)  
**subgoal for**  $x2$  **apply**(*cases x2*) **apply** (*auto simp add: u-defs K2exit-def*)  
**apply** (*metis less-not-refl paperIDs-equals reachNT-reach*) .  
**subgoal for**  $x3$  **apply**(*cases x3*) **apply** (*auto simp add: uu-defs K2exit-def*)  
**apply** (*metis isRev-def3 paperIDs-equals reachNT-reach*) .  
**by auto**  
**done**

**definition** *K3exit* **where**

$K3exit\ cid\ s \equiv PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > revPH$

**lemma** *invarNT-K3exit*: *invarNT (K3exit cid)*

**unfolding** *invarNT-def* **apply** (*safe dest!: reachNT-reach*)

**subgoal for** -  $a$  **apply**(*cases a*)

**subgoal for**  $x1$  **apply**(*cases x1*) **apply** (*fastforce simp add: c-defs K3exit-def*  
*geq-noPH-confIDs*) $+$  .

**subgoal for**  $x2$  **apply**(*cases x2*) **apply** (*fastforce simp add: u-defs K3exit-def*  
*paperIDs-equals*) $+$  .

**subgoal for**  $x3$  **apply**(*cases x3*) **apply** (*fastforce simp add: uu-defs K3exit-def*) $+$

.

**by auto**

**done**

**lemma** *noVal-K3exit*: *noVal (K3exit cid) (revPH,u)*

**unfolding** *noVal-def* **apply** *safe*

**subgoal for** -  $a$  **apply**(*cases a*)

**subgoal by** (*fastforce simp add: c-defs K3exit-def*)

**subgoal for**  $x2$  **apply**(*cases x2*) **apply** (*auto simp add: u-defs K3exit-def*)

**apply** (*metis less-not-refl paperIDs-equals reachNT-reach*) .

**subgoal for**  $x3$  **apply**(*cases x3*) **apply** (*auto simp add: uu-defs K3exit-def*) .

**by auto**

**done**

**lemma** *unwind-exit- $\Delta e$* : *unwind-exit  $\Delta e$*

**proof**

**fix**  $s\ s1 :: state$  **and**  $vl\ vl1 :: value\ list$

**assume**  $rsT: reachNT\ s$  **and**  $rs1: reach\ s1$  **and**  $\Delta e: \Delta e\ s\ vl\ s1\ vl1$

**hence**  $rs: reach\ s$  **and**  $vl: vl \neq []$  **using** *reachNT-reach* **unfolding**  *$\Delta e$ -def* **by**  
*auto*

**then obtain**  $CID$  **where**  $K: K2exit\ CID\ s \vee K3exit\ CID\ s$  **and**  $PID: PID \in \in$   
*paperIDs\ s\ CID*

**using**  *$\Delta e$*  **unfolding** *K2exit-def K3exit-def  $\Delta e$ -def* **by auto**

**show**  $vl \neq [] \wedge exit\ s\ (hd\ vl)$  **proof**(*simp add: vl, cases K2exit CID s*)

**case** *True*

**thus**  $exit\ s\ (hd\ vl)$

**by** (*metis rsT exitI2 invarNT-K2exit noVal-K2exit*)

**next**

**case** *False*

```

then obtain  $u$  where  $h: hd\ vl = (revPH, u)$  and  $K3: K3exit\ CID\ s$ 
using  $\Delta e\ K\ PID\ rs$  unfolding  $\Delta e\text{-def}\ K2exit\text{-def}\ K3exit\text{-def}$ 
by  $(cases\ vl)$   $(auto\ simp: isREVNth\text{-def})$ 
show  $exit\ s\ (hd\ vl)$  unfolding  $h$  using  $K3$ 
by  $(metis\ rsT\ exitI2\ invarNT\text{-}K3exit\ noVal\text{-}K3exit)$ 
qed
qed

```

```

theorem secure: secure
apply $(rule\ unwind\text{-}decomp4\text{-}secure[of\ \Delta 1\ \Delta 2\ \Delta e\ \Delta 3\ \Delta 4])$ 
using
 $istate\text{-}\Delta 1$ 
 $unwind\text{-}cont\text{-}\Delta 1\ unwind\text{-}cont\text{-}\Delta 2\ unwind\text{-}cont\text{-}\Delta 2\ unwind\text{-}cont\text{-}\Delta 3\ unwind\text{-}cont\text{-}\Delta 4$ 
 $unwind\text{-}exit\text{-}\Delta e$ 
by auto

```

```

end
theory Review-RAut-NCPC
imports ../Observation-Setup Review-Value-Setup Bounded-Deducibility-Security.Compositional-Reasoning
begin

```

## 6.4 Confidentiality protection from users who are not the review's author or a PC member

We verify the following property:

A group of users UIDs learn nothing about the various updates of the N'th review of a paper PID except for the last edited version before notification unless/until one of the following holds:

- a user in UIDs is the review's author, or
- a user in UIDs becomes a PC member in the paper's conference having no conflict with that paper, and the conference moves to the discussion phase.

```

type-synonym  $value = rcontent$ 

```

```

fun  $f :: (state, act, out)\ trans \Rightarrow value$  where
 $f\ (Trans\ -\ (Uact\ (uReview\ cid\ uid\ p\ pid\ n\ rc))\ -\ -) = rc$ 
 $|$ 
 $f\ (Trans\ -\ (UUact\ (uuReview\ cid\ uid\ p\ pid\ n\ rc))\ -\ -) = rc$ 

```

```

fun  $T :: (state, act, out)\ trans \Rightarrow bool$  where
 $T\ (Trans\ -\ -\ ou\ s') =$ 
 $(\exists\ uid \in UIDs.$ 
 $\quad isREVNth\ s'\ uid\ PID\ N$ 

```

$$\vee$$

$$(\exists \text{ cid. } PID \in \in \text{ paperIDs } s' \text{ cid} \wedge \text{ isPC } s' \text{ cid uid} \wedge \text{ pref } s' \text{ uid } PID \neq \text{ Conflict}$$

$$\wedge \text{ phase } s' \text{ cid} \geq \text{ disPH})$$

$$)$$

**declare**  $T.\text{simps}$  [ $\text{simp del}$ ]

**definition**  $B :: \text{value list} \Rightarrow \text{value list} \Rightarrow \text{bool}$  **where**  
 $B \text{ vl } vl1 \equiv \text{vl} \neq [] \wedge \text{vl1} \neq [] \wedge \text{last vl} = \text{last vl1}$

**interpretation**  $BD\text{-Security-IO}$  **where**  
 $\text{istate} = \text{istate}$  **and**  $\text{step} = \text{step}$  **and**  
 $\varphi = \varphi$  **and**  $f = f$  **and**  $\gamma = \gamma$  **and**  $g = g$  **and**  $T = T$  **and**  $B = B$   
**done**

**lemma**  $\text{reachNT-non-isRevNth-isPC-isChair}$ :

**assumes**  $\text{reachNT } s$  **and**  $\text{uid} \in \text{UIDs}$

**shows**

$\neg \text{isRevNth } s \text{ cid uid } PID \ N \wedge$   
 $(PID \in \in \text{ paperIDs } s \text{ cid} \wedge \text{ isPC } s \text{ cid uid} \longrightarrow \text{pref } s \text{ uid } PID = \text{ Conflict} \vee \text{ phase}$   
 $s \text{ cid} < \text{disPH}) \wedge$

$(PID \in \in \text{ paperIDs } s \text{ cid} \wedge \text{ isChair } s \text{ cid uid} \longrightarrow \text{pref } s \text{ uid } PID = \text{ Conflict} \vee$   
 $\text{phase } s \text{ cid} < \text{disPH})$

**using**  $\text{assms}$

**apply**  $\text{induct}$

**apply** ( $\text{auto simp: istate-def}$ )[]

**apply**( $\text{intro conjI}$ )

**subgoal for**  $\text{trn}$  **apply**( $\text{cases trn, auto simp: } T.\text{simps reachNT-reach isREVNth-def}$ )[]

.

**subgoal by** ( $\text{metis } T.\text{elims}(3) \text{ not-le-imp-less tgtOf-simps}$ )

**by** ( $\text{metis } T.\text{elims}(3) \text{ isChair-isPC not-le-imp-less reach.Step reachNT-reach tgtOf-simps}$ )

**lemma**  $T\text{-}\varphi\text{-}\gamma$ :

**assumes**  $1: \text{reachNT } s$  **and**  $2: \text{step } s \ a = (\text{ou}, s')$   $\varphi (\text{Trans } s \ a \ \text{ou } s')$

**shows**  $\neg \gamma (\text{Trans } s \ a \ \text{ou } s')$

**using**  $\text{reachNT-non-isRevNth-isPC-isChair}[OF \ 1] \ 2$  **unfolding**  $T.\text{simps } \varphi\text{-def2}$

**apply** ( $\text{auto simp add: u-defs uu-defs}$ ) **by** ( $\text{metis isRev-imp-isRevNth-getReviewIndex}$ )+

**lemma**  $\text{eqExcPID-N-step-out}$ :

**assumes**  $s's1'$ :  $\text{eqExcPID-N } s \ s1$

**and**  $\text{step: step } s \ a = (\text{ou}, s')$  **and**  $\text{step1: step } s1 \ a = (\text{ou1}, s1')$

**and**  $sP: \text{reachNT } s$  **and**  $s1: \text{reach } s1$

**and**  $PID: PID \in \in \text{ paperIDs } s \ \text{cid}$

**and**  $ph: \text{phase } s \ \text{cid} = \text{revPH} \vee \text{phase } s \ \text{cid} = \text{disPH}$

**and**  $\text{UIDs: userOfA } a \in \text{UIDs}$

**shows**  $\text{ou} = \text{ou1}$

**proof** –

**note**  $\text{Inv} = \text{reachNT-non-isRevNth-isPC-isChair}[OF \ sP \ \text{UIDs}]$

**note**  $\text{eqs} = \text{eqExcPID-N-imp}[OF \ s's1']$

```

note eqs' = eqExcPID-N-imp1[OF s's1']
note eqss = eqExcPID-N-imp2[OF s's1']
note s = reachNT-reach[OF sP]

note_simps[simp] = c-defs u-defs uu-defs r-defs l-defs Paper-dest-conv eqExc-
cPID-N-def eqExcPID-N-def eqExcD
note * = step step1 eqs eqs' s s1 PID UIDs ph paperIDs-equals[OF s] Inv

show ?thesis
proof (cases a)
  case (Cact x1)
    with * show ?thesis by (cases x1; auto)
  next
    case (Uact x2)
      with * show ?thesis by (cases x2; auto)
    next
      case (UUact x3)
        with * show ?thesis by (cases x3; auto)
      next
        case (Ract x4)
          with * show ?thesis
          proof (cases x4)
            case (rMyReview x81 x82 x83 x84)
              with Ract * show ?thesis
              by clarsimp (metis eqExcPID-N-imp3' getRevRole-Some-Rev-isRevNth s's1')
            next
              case (rReviews x91 x92 x93 x94)
                with Ract * show ?thesis
                by clarsimp (metis eqss not-less)
              next
                case (rFinalReviews x121 x122 x123 x124)
                  with Ract * show ?thesis
                  by clarsimp (metis Suc-leD Suc-n-not-le-n)
                qed auto
              next
                case (Lact x5)
                  with * show ?thesis by (cases x5; auto; presburger)
                qed
              qed
            qed
          qed
        next
          case (Lact x5)
            with * show ?thesis by (cases x5; auto; presburger)
          qed
        qed
      qed
    qed
  qed

```

**lemma** eqExcPID-N2-step-out:  
**assumes** ss1: eqExcPID-N2 s s1  
**and** step: step s a = (ou,s') **and** step1: step s1 a = (ou1,s1')  
**and** sP: reachNT s **and** s1: reach s1  
**and** r: isRevNth s cid uid PID N  
**and** ph: phase s cid ≥ revPH  
**and** UIDs: userOfA a ∈ UIDs  
**and** decs-exit: (reviewsPaper (paper s PID))!N ≠ [] ∧ (reviewsPaper (paper s1 PID))!N ≠ []

```

shows  $ou = ou1$ 
proof –
  note  $s = reachNT-reach[OF sP]$ 
  note  $Inv = reachNT-non-isRevNth-isPC-isChair[OF sP UIDs]$ 
  note  $eqs = eqExcPID-N2-imp[OF ss1]$ 
  note  $eqs' = eqExcPID-N2-imp1[OF ss1]$ 
  note  $eqss = eqExcPID-N2-imp2[OF ss1] eqExcPID-N2-imp3'[OF s ss1] eqExcPID-N2-imp33[OF ss1]$ 

  have  $PID: PID \in \in paperIDs s cid$  using  $r$  by  $(metis isRevNth-paperIDs s)$ 
  have  $PID1: PID \in \in paperIDs s1 cid$  using  $PID ss1$  unfolding  $eqExcPID-N2-def$ 
by  $auto$ 
  have  $r1: isRevNth s1 cid uid PID N$  by  $(metis eqs r)$ 
  hence  $decs-exit': (reviewsPaper (paper s' PID))!N \neq [] \wedge$ 
     $(reviewsPaper (paper s1' PID))!N \neq []$ 
  using  $nonempty-reviews-persist s s1 PID PID1 r r1 decs-exit step step1$  by
 $metis+$ 

  note  $simps[simp] = c-defs u-defs uu-defs r-defs l-defs Paper-dest-conv eqExcPID-N2-def eeqExcPID-N2-def eqExcD2$ 

  have  $eqExcD2 (paper s PID) (paper s1 PID)$ 
  using  $eqExcPID-N2-imp[OF ss1] eeqExcPID-N2-imp$  by  $blast$ 
  hence  $1: hd (reviewsPaper (paper s PID) ! N) =$ 
     $hd (reviewsPaper (paper s1 PID) ! N)$ 
  unfolding  $eqExcD2 eqExcNth2-def$  by  $auto$ 

  { fix  $cid uid p pid$  assume  $a: a = Ract (rFinalReviews cid uid p pid)$ 
    have  $?thesis$  using  $step step1 eqExcPID-N2-imp[OF ss1]$ 
    unfolding  $a$ 
    apply  $clarsimp$ 
    apply  $(intro nth-equalityI)$ 
    subgoal by  $simp$ 
    subgoal for  $i$  apply  $(cases i \neq N)$ 
      subgoal by  $smt (verit) eqExcPID-N2-imp3 getNthReview-def ss1)$ 
      by  $(auto split: list.splits)$ 
    subgoal for  $i ia$ 
      apply  $(cases pid = PID)$ 
      subgoal
        apply  $(cases reviewsPaper (paper s' PID) ! i)$ 
        subgoal apply  $simp$ 
        by  $(smt (verit) decs-exit eqExcPID-N2-imp3 getNthReview-def list.simps(4) nth-Cons-0 ss1)$ 
        subgoal apply  $(cases reviewsPaper (paper s1' PID) ! i)$ 
        subgoal apply  $simp$ 
        by  $(metis (no-types, lifting) decs-exit eqExcD2 eqExcNth2-def neq-Nil-conv)$ 
        subgoal apply  $simp$ 
        by  $(metis (no-types, lifting) eqExcD2 eqExcNth2-def list.sel(1)) . .$ 
        subgoal by  $simp . .$ 

```

```

} note this[simp]

note * = step step1 eqs eqs' s s1 PID PID1 r r1 UIDs ph paperIDs-equals[OF s]
Inv

show ?thesis
proof (cases a)
  case (Cact x1)
  with * show ?thesis by (cases x1; auto)
next
  case (Uact x2)
  with * show ?thesis by (cases x2; auto)
next
  case (UUact x3)
  with * show ?thesis by (cases x3; auto)
next
  case (Ract x4)
  with * show ?thesis
  proof (cases x4)
    case (rMyReview x81 x82 x83 x84)
    with Ract * show ?thesis
    by clarsimp (metis eqss(2) getRevRole-Some-Rev-isRevNth)
  next
    case (rReviews x91 x92 x93 x94)
    with Ract * show ?thesis
    by clarsimp (metis eqss(1) not-less)
  qed auto
next
  case (Lact x5)
  with * show ?thesis by (cases x5; auto; presburger)
qed
qed

lemma eqExcPID-N-step-eqExcPID-N2:
assumes rs: reach s
and a: a = Uact (uReview cid uid p PID N rc) ∨
      a = UUact (uuReview cid uid p PID N rc) (is ?L ∨ ?R)
and ss1: eqExcPID-N s s1
and step: step s a = (outOK,s') and step1: step s1 a = (outOK,s1')
shows eqExcPID-N2 s' s1'
using a proof
  assume a: ?L
  have isRevNth s cid uid PID N using step unfolding a apply(simp add: u-defs
uu-defs)
  by (metis isRev-imp-isRevNth-getReviewIndex)
  hence N: N < length (reviewsPaper (paper s PID))
  using rs by (metis isRevNth-less-length)
  hence N1: N < length (reviewsPaper (paper s1 PID))
  using ss1 unfolding eqExcPID-N-def eqExcPID-N-def eqExcD eqExcNth-def

```

**by** *auto*  
**have** *eqExcPID-N s' s1'* **using** *assms* **by** (*metis eqExcPID-N-step*)  
**moreover have** *hd (reviewsPaper (paper s' PID) ! N) = hd (reviewsPaper (paper s1' PID) ! N)*  
**using** *step step1 N N1* **unfolding** *a* **by**(*auto simp add: u-defs uu-defs*)  
**ultimately show** *?thesis*  
**unfolding** *eqExcPID-N-def eqExcPID-N2-def eeqExcPID-N-def eeqExcPID-N2-def eqExcD2 eqExcD eqExcNth-def eqExcNth2-def* **by** *auto*  
**next**  
**assume** *a: ?R*  
**have** *isRevNth s cid uid PID N* **using** *step* **unfolding** *a* **apply**(*simp add: u-defs uu-defs*)  
**by** (*metis isRev-imp-isRevNth-getReviewIndex*)  
**hence** *N: N < length (reviewsPaper (paper s PID))*  
**using** *rs* **by** (*metis isRevNth-less-length*)  
**hence** *N1: N < length (reviewsPaper (paper s1 PID))*  
**using** *ss1* **unfolding** *eqExcPID-N-def eeqExcPID-N-def eqExcD eqExcNth-def*  
**by** *auto*  
**have** *eqExcPID-N s' s1'* **using** *assms* **by** (*metis eqExcPID-N-step*)  
**moreover have** *hd (reviewsPaper (paper s' PID) ! N) = hd (reviewsPaper (paper s1' PID) ! N)*  
**using** *step step1 N N1* **unfolding** *a* **by**(*auto simp add: u-defs uu-defs*)  
**ultimately show** *?thesis*  
**unfolding** *eqExcPID-N-def eqExcPID-N2-def eeqExcPID-N-def eeqExcPID-N2-def eqExcD2 eqExcD eqExcNth-def eqExcNth2-def* **by** *auto*  
**qed**

**lemma** *eqExcPID-N-step-φ-eqExcPID-N2*:  
**assumes** *rs: reach s*  
**and** *ss1: eqExcPID-N s s1*  
**and** *step: step s a = (ou, s')* **and** *step1: step s1 a = (ou1, s1')*  
**and** *φ: φ (Trans s a ou s')*  
**shows** *eqExcPID-N2 s' s1'*  
**proof**–  
**obtain** *cid uid p rc* **where**  
*a: a = Uact (uReview cid uid p PID N rc) ∨*  
*a = UUact (uuReview cid uid p PID N rc) (is ?L ∨ ?R)*  
**and** *ou: ou = outOK*  
**using** *φ* **unfolding** *φ-def2* **by** *blast*  
**have** *φ1: φ (Trans s1 a ou1 s1')* **using** *φ ss1* **by** (*metis eqExcPID-N-step-φ-imp step step1*)  
**hence** *ou1: ou1 = outOK* **using** *φ* **unfolding** *φ-def2* **by** *auto*  
**show** *?thesis* **using** *eqExcPID-N-step-eqExcPID-N2[OF rs a ss1 step[unfolded ou] step1[unfolded ou1]]*.  
**qed**

**definition**  $\Delta 1 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**

$\Delta 1$   $s$   $vl$   $s1$   $vl1$   $\equiv$   
 $(\forall cid. PID \in \in paperIDs\ s\ cid \longrightarrow phase\ s\ cid < revPH) \wedge$   
 $s = s1 \wedge B\ vl\ vl1$

**definition**  $\Delta 2$   $:: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 2$   $s$   $vl$   $s1$   $vl1$   $\equiv$   
 $\exists cid.$   
 $PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid = revPH \wedge \neg (\exists uid. isREVNth\ s\ uid$   
 $PID\ N) \wedge$   
 $s = s1 \wedge B\ vl\ vl1$

**definition**  $\Delta 3$   $:: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 3$   $s$   $vl$   $s1$   $vl1$   $\equiv$   
 $\exists cid\ uid.$   
 $PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid \in \{revPH, disPH\} \wedge isREVNth\ s\ uid$   
 $PID\ N \wedge$   
 $eqExcPID-N\ s\ s1 \wedge B\ vl\ vl1$

**definition**  $\Delta 4$   $:: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 4$   $s$   $vl$   $s1$   $vl1$   $\equiv$   
 $\exists cid\ uid.$   
 $PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid \geq revPH \wedge isREVNth\ s\ uid\ PID\ N \wedge$   
 $(reviewsPaper\ (paper\ s\ PID))!N \neq [] \wedge (reviewsPaper\ (paper\ s1\ PID))!N \neq []$   
 $\wedge$   
 $eqExcPID-N2\ s\ s1 \wedge vl = [] \wedge vl1 = []$

**definition**  $\Delta e$   $:: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta e$   $s$   $vl$   $s1$   $vl1$   $\equiv$   
 $vl \neq [] \wedge$   
 $($   
 $(\exists cid. PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > revPH \wedge \neg (\exists uid. isREVNth\ s$   
 $uid\ PID\ N))$   
 $\vee$   
 $(\exists cid. PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > disPH)$   
 $)$

**lemma** *istate- $\Delta 1$* :  
**assumes**  $B$ :  $B\ vl\ vl1$   
**shows**  $\Delta 1$  *istate*  $vl$  *istate*  $vl1$   
**using**  $B$  **unfolding**  $\Delta 1$ -def  $B$ -def *istate*-def **by** *auto*

**lemma** *unwind-cont- $\Delta 1$* : *unwind-cont*  $\Delta 1$   $\{\Delta 1, \Delta 2, \Delta e\}$   
**proof**(*rule, simp*)  
**let**  $?\Delta = \lambda s\ vl\ s1\ vl1. \Delta 1\ s\ vl\ s1\ vl1 \vee \Delta 2\ s\ vl\ s1\ vl1 \vee \Delta e\ s\ vl\ s1\ vl1$   
**fix**  $s\ s1$   $:: state$  **and**  $vl\ vl1$   $:: value\ list$   
**assume**  $rsP$ : *reachNT*  $s$  **and**  $rs1$ : *reach*  $s1$  **and**  $\Delta 1\ s\ vl\ s1\ vl1$   
**hence**  $rs$ : *reach*  $s$  **and**  $ss1$ :  $s1 = s$   
**and**  $vl$ :  $vl \neq []$  **and**  $vl1$ :  $vl1 \neq []$  **and**  $vl$ - $vl1$ : *last*  $vl1 = last\ vl$   
**and**  $PID$ - $ph$ :  $\bigwedge cid. PID \in \in paperIDs\ s\ cid \implies phase\ s\ cid < revPH$

```

using reachNT-reach unfolding  $\Delta 1$ -def B-def by auto
note vlvl1 = vl vl1 vl-vl1
show iaction ? $\Delta$  s vl s1 vl1  $\vee$ 
  ((vl = []  $\longrightarrow$  vl1 = [])  $\wedge$  reaction ? $\Delta$  s vl s1 vl1) (is ?iact  $\vee$  (-  $\wedge$  ?react))
proof-
  have ?react proof
    fix a :: act and ou :: out and s' and vl'
    let ?trn = Trans s a ou s' let ?trn1 = Trans s1 a ou s'
    assume step: step s a = (ou, s') and P:  $\neg T$  ?trn and c: consume ?trn vl vl'
    have  $\varphi$ :  $\neg \varphi$  ?trn
    apply(cases a)
    subgoal by simp
    subgoal for x2 apply(cases x2) using step PID-ph by (fastforce simp:
u-defs)+
    subgoal for x3 apply(cases x3) using step PID-ph by (fastforce simp:
uu-defs)+
    by simp-all
    hence vl': vl' = vl using c unfolding consume-def by auto
    show match ? $\Delta$  s s1 vl1 a ou s' vl'  $\vee$  ignore ? $\Delta$  s s1 vl1 a ou s' vl' (is ?match
 $\vee$  ?ignore)
    proof-
      have ?match proof
        show validTrans ?trn1 unfolding ss1 using step by simp
        next
        show consume ?trn1 vl1 vl1 unfolding consume-def ss1 using  $\varphi$  by auto
        next
        show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
        next
        assume  $\gamma$  ?trn thus g ?trn = g ?trn1 unfolding ss1 by simp
        next
        show ? $\Delta$  s' vl' s' vl1
        proof(cases  $\exists$  cid. PID  $\in \in$  paperIDs s cid)
          case False note PID = False
          have PID-ph':  $\bigwedge$  cid. PID  $\in \in$  paperIDs s' cid  $\implies$  phase s' cid < revPH
using PID step rs
          subgoal apply(cases a)
            subgoal for x1 apply(cases x1) apply(fastforce simp: c-defs)+ .
            subgoal for x2 apply(cases x2) apply(fastforce simp: u-defs)+ .
            subgoal for x3 apply(cases x3) apply(fastforce simp: uu-defs)+ .
            by auto
          done
          hence  $\Delta 1$  s' vl' s' vl1 unfolding  $\Delta 1$ -def B-def B-def vl' using PID-ph'
vlvl1 by auto
          thus ?thesis by auto
        next
        case True
        then obtain CID where PID: PID  $\in \in$  paperIDs s CID by auto
        hence ph: phase s CID < revPH using PID-ph by auto
        have PID': PID  $\in \in$  paperIDs s' CID by (metis PID paperIDs-mono

```

```

step)
  show ?thesis
  proof(cases phase s' CID < revPH)
    case True note ph' = True
      hence  $\Delta 1$  s' vl' s' vl1 unfolding  $\Delta 1$ -def B-def B-def vl' using vlvl1
  ph' PID' apply auto
    by (metis reach-PairI paperIDs-equals rs step)
    thus ?thesis by auto
  next
    case False note ph' = False
      have  $\neg (\exists uid. isRevNth s CID uid PID N)$  using rs ph is-
  RevNth-geq-revPH by fastforce
      hence ph-isRev': phase s' CID = revPH  $\wedge \neg (\exists uid. isRevNth s' CID$ 
  uid PID N)
      using ph' ph PID step rs
      subgoal apply(cases a)
        subgoal for x1 apply(cases x1) apply(fastforce simp: c-defs)+ .
        subgoal for x2 apply(cases x2) apply(fastforce simp: u-defs)+ .
        subgoal for x3 apply(cases x3) apply(fastforce simp: uu-defs)+ .
        by auto
      done
      hence  $\neg (\exists uid. isREVNth s' uid PID N)$ 
      by (metis PID' isREVNth-imp-isRevNth reach-PairI rs step)
      hence  $\Delta 2$  s' vl' s' vl1 unfolding  $\Delta 2$ -def B-def vl' using vlvl1 ph' PID'
  ph-isRev' by auto
      thus ?thesis by auto
    qed
  qed
  qed
  thus ?thesis by simp
  qed
  qed
  thus ?thesis using vl by auto
  qed
  qed

```

```

lemma unwind-cont- $\Delta 2$ : unwind-cont  $\Delta 2$  { $\Delta 2, \Delta 3, \Delta e$ }
proof(rule, simp)
  let ? $\Delta$  =  $\lambda s vl s1 vl1. \Delta 2 s vl s1 vl1 \vee \Delta 3 s vl s1 vl1 \vee \Delta e s vl s1 vl1$ 
  fix s s1 :: state and vl vl1 :: value list
  assume rsP: reachNT s and rs1: reach s1 and  $\Delta 2 s vl s1 vl1$ 
  then obtain CID where rs: reach s and ph: phase s CID = revPH (is ?ph =
-) and ss1: s1 = s
  and uvid:  $\neg (\exists uid. isREVNth s uid PID N)$ 
  and vl: vl  $\neq []$  and vl1: vl1  $\neq []$  and vl-vl1: last vl1 = last vl
  and PID: PID  $\in \in$  paperIDs s CID using reachNT-reach unfolding  $\Delta 2$ -def B-def
  by auto
  hence uid:  $\neg (\exists uid. isRevNth s CID uid PID N)$  by (metis isREVNth-def)
  note vlvl1 = vl vl1 vl-vl1

```

```

show iaction ? $\Delta$  s vl s1 vl1  $\vee$ 
  ((vl = []  $\longrightarrow$  vl1 = [])  $\wedge$  reaction ? $\Delta$  s vl s1 vl1) (is ?iact  $\vee$  ( $\neg$   $\wedge$  ?react))
proof–
  have ?react proof
    fix a :: act and ou :: out and s' :: state and vl'
    let ?trn = Trans s a ou s' let ?trn1 = Trans s1 a ou s'
    let ?ph' = phase s' CID
    assume step: step s a = (ou, s') and P:  $\neg$  T ?trn and c: consume ?trn vl vl'
    have  $\varphi$ :  $\neg$   $\varphi$  ?trn
      apply(cases a)
      subgoal by simp
      subgoal for x2 apply(cases x2)
        using step ph apply (auto simp: u-defs)
        by (metis PID isRev-imp-isRevNth-getReviewIndex paperIDs-equals rs uid)
      subgoal for x3 apply(cases x3)
        using step ph apply (auto simp: uu-defs)
        using PID paperIDs-equals rs by force
      by simp-all
    hence vl': vl' = vl using c unfolding consume-def by auto
    have PID': PID  $\in\in$  paperIDs s' CID by (metis paperIDs-mono step PID)
    show match ? $\Delta$  s s1 vl1 a ou s' vl'  $\vee$  ignore ? $\Delta$  s s1 vl1 a ou s' vl' (is ?match
 $\vee$  ?ignore)
    proof–
      have ?match proof
        show validTrans ?trn1 unfolding ss1 using step by simp
        next
        show consume ?trn1 vl1 vl1 unfolding consume-def ss1 using  $\varphi$  by auto
        next
        show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
        next
        assume  $\gamma$  ?trn thus g ?trn = g ?trn1 unfolding ss1 by simp
        next
        show ? $\Delta$  s' vl' s' vl1
        proof(cases ?ph' = revPH)
          case False
            hence 1: ?ph' > revPH  $\wedge$   $\neg$  ( $\exists$  uid. isRevNth s' CID uid PID N)
            using uid PID ph step rs
            subgoal apply(cases a)
              subgoal for x1 apply(cases x1) apply(fastforce simp: c-defs) $+$  .
              subgoal for x2 apply(cases x2) apply(fastforce simp: u-defs) $+$  .
              subgoal for x3 apply(cases x3) apply(fastforce simp: uu-defs) $+$  .
              by auto
            done
            hence  $\neg$  ( $\exists$  uid. isREVNth s' uid PID N)
            by (metis PID' isREVNth-imp-isRevNth reach-PairI rs step)
            hence  $\Delta$  e s' vl' s' vl1 unfolding  $\Delta$ -def vl' using PID' vl 1 by auto
            thus ?thesis by simp
          case True note ph' = True
        next
        case True note ph' = True

```

```

      show ?thesis proof(cases  $\exists uid. isREVNth s' uid PID N$ )
        case False
          hence  $\Delta 2 s' vl' s' vl1$  using PID' vlvl1 ph' unfolding  $\Delta 2$ -def B-def
    vl' by auto
      thus ?thesis by simp
    next
      case True
          hence  $\Delta 3 s' vl' s' vl1$  using PID' vlvl1 ph' unfolding  $\Delta 3$ -def B-def
    vl' by auto
      thus ?thesis by simp
    qed
  qed
  qed
  thus ?thesis by simp
  qed
  thus ?thesis using vl by auto
  qed
  qed

```

```

lemma unwind-cont- $\Delta 3$ : unwind-cont  $\Delta 3 \{\Delta 3, \Delta 4, \Delta e\}$ 
proof(rule, simp)
  let ? $\Delta = \lambda s vl s1 vl1. \Delta 3 s vl s1 vl1 \vee \Delta 4 s vl s1 vl1 \vee \Delta e s vl s1 vl1$ 
  fix s s1 :: state and vl vl1 :: value list
  assume rsT: reachNT s and rs1: reach s1 and  $\Delta 3 s vl s1 vl1$ 
  then obtain CID uid where uuid: isREVNth s uid PID N
  and PID: PID  $\in \in$  paperIDs s CID
  and rs: reach s and ph: phase s CID = revPH  $\vee$  phase s CID = disPH (is ?ph
= -  $\vee$  -)
  and ss1: eqExcPID-N s s1 and vl: vl  $\neq$  [] and vl1: vl1  $\neq$  [] and vl-vl1: last vl
= last vl1
  using reachNT-reach unfolding  $\Delta 3$ -def B-def by auto
  hence uid: isRevNth s CID uid PID N by (metis isREVNth-imp-isRevNth)
  note vlvl1 = vl vl1 vl-vl1
  from vl vl1 obtain v vl' v1 vl1' where vl: vl = v # vl' and vl1: vl1 = v1 #
vl1' by (metis list.exhaust)
  have uid-notin: uid  $\notin$  UIDs using uid by (metis reachNT-non-isRevNth-isPC-isChair
rsT)
  show iaction ? $\Delta s vl s1 vl1 \vee$ 
    (vl = []  $\longrightarrow$  vl1 = [])  $\wedge$  reaction ? $\Delta s vl s1 vl1$  (is ?iact  $\vee$  (-  $\wedge$  ?react))
  proof(cases vl1' = [])
    case False note vl1' = False
      hence vl-vl1': last vl = last vl1' using vl-vl1 unfolding vl1 by simp
      have uid1: isRevNth s CID uid PID N using ss1 uid unfolding eqExcPID-N-def
by auto
      define a1 where a1  $\equiv$ 
        if ?ph = revPH
        then Uact (uReview CID uid (pass s uid) PID N v1)
        else UUact (uuReview CID uid (pass s uid) PID N v1)

```

```

(is - ≡ if ?ph = revPH then ?A else ?B)
hence a1: a1 ∈ {?A,?B} by auto
obtain s1' ou1 where step1: step s1 a1 = (ou1,s1') by (metis prod.exhaust)
let ?trn1 = Trans s1 a1 ou1 s1'
have s1s1': eqExcPID-N s1 s1' using step1 by (metis a1-def uReview-uuReview-step-eqExcPID-N)
have ss1': eqExcPID-N s s1' using eqExcPID-N-trans[OF ss1 s1s1'] .
hence many-s1': PID ∈∈ paperIDs s1' CID isRevNth s1' CID uid PID N
pass s1' uid = pass s uid phase s1' CID = phase s CID
using uid PID ph unfolding eqExcPID-N-def by simp-all
hence more-s1': uid ∈∈ userIDs s1' CID ∈∈ confIDs s1'
by (metis paperIDs-confIDs reach-PairI roles-userIDs rs1 step1 many-s1'(1))+
have f: f ?trn1 = v1 unfolding a1-def by simp
have rs1': reach s1' using rs1 step1 by (auto intro: reach-PairI)
have ou1: ou1 = outOK
using step1 uid1 ph unfolding a1-def apply (simp-all add: u-defs uu-defs
many-s1' more-s1')
by (metis isRevNth-getReviewIndex isRev-def3 many-s1' rs1')+
have ?iact proof
show step s1 a1 = (ou1,s1') by fact
next
show φ: φ ?trn1 using ou1 unfolding a1-def by simp
thus consume ?trn1 vl1 vl1' using f unfolding consume-def vl1 by simp
next
show ¬ γ ?trn1 by (simp add: a1-def uid-notin)
next
have Δ3 s vl s1' vl1' unfolding Δ3-def B-def using ph PID ss1' uuid vl-vl1'
vl1' vl by auto
thus ?Δ s vl s1' vl1' by simp
qed
thus ?thesis by auto
next
case True hence vl1: vl1 = [v1] unfolding vl1 by simp
have ?react proof
fix a :: act and ou :: out and s' :: state and vll'
let ?trn = Trans s a ou s'
let ?ph' = phase s' CID
assume step: step s a = (ou, s') and T: ¬ T ?trn and c: consume ?trn vl
vll'
have PID': PID ∈∈ paperIDs s' CID using PID rs by (metis paperIDs-mono
step)
have uid': isRevNth s' CID uid PID N using uid step rs ph PID is-
RevNth-persistent by auto
hence uuid': isREVNth s' uid PID N by (metis isREVNth-def)
show match ?Δ s s1 vl1 a ou s' vll' ∨ ignore ?Δ s s1 vl1 a ou s' vll' (is
?match ∨ ?ignore)
proof(cases φ ?trn)
case False note φ = False
have vll': vll' = vl using c φ unfolding consume-def by (cases vl) auto
obtain ou1 and s1' where step1: step s1 a = (ou1,s1') by (metis

```

```

prod.exhaust)
  let ?trn1 = Trans s1 a ou1 s1'
  have s's1': eqExcPID-N s' s1' using eqExcPID-N-step[OF ss1 step step1] .
  have  $\varphi 1$ :  $\neg \varphi$  ?trn1 using  $\varphi$  unfolding eqExcPID-N-step- $\varphi$ [OF ss1 step
step1] .
  have ?match proof
    show validTrans ?trn1 using step1 by simp
  next
    show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi 1$  by auto
  next
    show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
  next
    assume  $\gamma$  ?trn thus  $g$  ?trn =  $g$  ?trn1
    using eqExcPID-N-step-out[OF ss1 step step1 rsT rs1 PID ph] by simp
  next
    show  $\Delta$  s' vll' s1' vl1
    proof(cases ?ph' = revPH  $\vee$  ?ph' = disPH)
      case True
        hence  $\Delta 3$  s' vll' s1' vl1 using PID' s's1' uuid' vlvl1 unfolding  $\Delta 3$ -def
B-def vll' by auto
        thus ?thesis by auto
      next
        case False hence ph': ?ph' > disPH using ph rs step
        by (metis le-less less-antisym not-less phase-increases2 prod.sel(2))
        hence  $\Delta e$  s' vll' s1' vl1 unfolding  $\Delta e$ -def vll' using PID' vlvl1 by auto
        thus ?thesis by auto
    qed
  qed
  thus ?thesis by simp
next
case True note  $\varphi = \text{True}$ 
hence vll': vll' = vl' using c unfolding vl consume-def by simp
obtain cid uid p rc where a:
a = Uact (uReview cid uid p PID N rc)  $\vee$ 
a = UUact (uuReview cid uid p PID N rc) (is a = ?A  $\vee$  a = ?B)
and ou: ou = outOK and v: v = rc
using  $\varphi$  c unfolding vl consume-def  $\varphi$ -def2 vll' by fastforce
hence cid: cid = CID using step apply(auto simp: u-defs uu-defs)
by (metis PID paperIDs-equals rs)+
have a: (?ph = revPH  $\longrightarrow$  a = ?A)  $\wedge$  (?ph = disPH  $\longrightarrow$  a = ?B)
using step ou a by (cases a = ?A, auto simp: u-defs uu-defs cid)
have  $\gamma$ :  $\neg \gamma$  ?trn using step T rsT by (metis T- $\varphi$ - $\gamma$  True)
hence f: f ?trn = v using c  $\varphi$  unfolding consume-def vl by auto
have s's: eqExcPID-N s' s using eqExcPID-N-sym[OF  $\varphi$ -step-eqExcPID-N[OF
 $\varphi$  step]] .
have s's1: eqExcPID-N s' s1 using eqExcPID-N-trans[OF s's ss1] .
have ph': phase s' CID = ?ph using s's ph unfolding eqExcPID-N-def by
auto
show ?thesis

```

```

proof(cases vl' = [])
  case False note vl' = False
  hence vl'-vl1: last vl' = last vl1 using vl-vl1 unfolding vl by auto
  have ?ignore proof
    show ¬ γ ?trn by fact
  next
    show ?Δ s' vll' s1 vl1
    proof(cases ?ph' = revPH ∨ ?ph' = disPH)
      case True
        hence Δ3 s' vll' s1 vl1 using s's1 PID' uuid' vl' vl1 vl-vl1 unfolding
Δ3-def B-def vl vll' by auto
        thus ?thesis by auto
      next
        case False hence ?ph' > disPH using ph rs step by (simp add: ph')
        hence Δe s' vll' s1 vl1 unfolding Δe-def vll' using PID' vvl1 vl' by
auto
        thus ?thesis by auto
    qed
  qed
  thus ?thesis by auto
next
  case True note vl' = True hence vl: vl = [v] unfolding vl by simp

  hence v1v: v1 = v using vl-vl1 unfolding vl1 by simp
obtain s1' ou1 where step1: step s1 a = (ou1, s1') by (metis prod.exhaust)
let ?trn1 = Trans s1 a ou1 s1'
  have φ1: φ ?trn1 using eqExcPID-N-step-φ-imp[OF ss1 step step1 φ] .
  hence ou1: ou1 = outOK unfolding φ-def2 by auto
  have uid'-uid1': isRevNth s' CID uid PID N ∧ isRevNth s1' CID uid PID
N
  using step step1 ou ou1 ph a apply(auto simp: u-defs uu-defs)
  by (metis cid isRev-imp-isRevNth-getReviewIndex)+
  hence N: N < length (reviewsPaper (paper s' PID)) ∧ N < length
(reviewsPaper (paper s1' PID))
  by (metis isRevNth-less-length reach-PairI rs rs1 step step1)
  hence l: reviewsPaper (paper s' PID) ! N ≠ [] ∧ reviewsPaper (paper s1'
PID) ! N ≠ []
  using step step1 ph a ou ou1 by (auto simp add: u-defs uu-defs)
  have ?match proof
    show validTrans ?trn1 using step1 by simp
  next
    show consume ?trn1 vl1 [] unfolding consume-def using φ1 a ph
by (auto simp add: a v vl1 v1v)
  next
    show γ ?trn = γ ?trn1 unfolding ss1 by simp
  next
    assume γ ?trn thus g ?trn = g ?trn1
    using eqExcPID-N-step-out[OF ss1 step step1 rsT rs1 PID ph] by simp
  next

```

```

      have  $\Delta_4$  s' vll' s1' [] unfolding vll' vl'  $\Delta_4$ -def
      using ph' ph uuid' l eqExcPID-N-step- $\varphi$ -eqExcPID-N2[OF rs ss1 step step1
 $\varphi$ ] PID' by auto
      thus ? $\Delta$  s' vll' s1' [] by simp
    qed
    thus ?thesis by simp
  qed
  qed
  thus ?thesis using vl by auto
  qed
  qed

```

```

lemma unwind-cont- $\Delta_4$ : unwind-cont  $\Delta_4$  { $\Delta_4, \Delta e$ }
proof(rule, simp)
  let ? $\Delta$  =  $\lambda s$  vl s1 vl1.  $\Delta_4$  s vl s1 vl1  $\vee$   $\Delta e$  s vl s1 vl1
  fix s s1 :: state and vl vl1 :: value list
  assume rsT: reachNT s and rs1: reach s1 and  $\Delta_4$  s vl s1 vl1
  then obtain CID uid where uuid: isREVNth s uid PID N
  and rs: reach s and ph: phase s CID  $\geq$  revPH (is ?ph  $\geq$  -)
  and PID: PID  $\in$  paperIDs s CID
  and decs: (reviewsPaper (paper s PID))!N  $\neq$  []  $\wedge$  (reviewsPaper (paper s1 PID))!N
 $\neq$  []
  and ss1: eqExcPID-N2 s s1 and vl: vl = [] and vl1: vl1 = []
  using reachNT-reach unfolding  $\Delta_4$ -def by auto
  hence uid: isRevNth s CID uid PID N by (metis isREVNth-imp-isRevNth)
  show iaction ? $\Delta$  s vl s1 vl1  $\vee$ 
    ((vl = []  $\longrightarrow$  vl1 = [])  $\wedge$  reaction ? $\Delta$  s vl s1 vl1) (is ?iact  $\vee$  (-  $\wedge$  ?react))
  proof-
    have ?react
    proof
      fix a :: act and ou :: out and s' :: state and vl'
      let ?trn = Trans s a ou s'
      let ?ph' = phase s' CID
      assume step: step s a = (ou, s') and T:  $\neg$  T ?trn and c: consume ?trn vl vl'
      have ph': phase s' CID  $\geq$  revPH using ph rs isRevNth-geq-revPH is-
      RevNth-persistent local.step reach-PairI uid by blast
      have PID': PID  $\in$  paperIDs s' CID by (metis PID paperIDs-mono step)
      have uid': isRevNth s' CID uid PID N using isRevNth-persistent by (metis
      isRevNth-persistent rs step uid)
      hence uuid': isREVNth s' uid PID N by (metis isREVNth-def)
      show match ? $\Delta$  s s1 vl1 a ou s' vl'  $\vee$  ignore ? $\Delta$  s s1 vl1 a ou s' vl' (is ?match
 $\vee$  ?ignore)
    proof-
      have  $\varphi$ :  $\neg$   $\varphi$  ?trn and vl': vl' = [] using c unfolding consume-def vl by
      auto
      obtain ou1 and s1' where step1: step s1 a = (ou1, s1') by (metis
      prod.exhaust)
      let ?trn1 = Trans s1 a ou1 s1'

```

```

      have s's1': eqExcPID-N2 s' s1' using eqExcPID-N2-step[OF ss1 step step1
rs uid] .
      have  $\varphi 1$ :  $\neg \varphi$  ?trn1 using  $\varphi$  unfolding eqExcPID-N2-step- $\varphi$ [OF rs rs1 ss1
step step1] .
      have uid1: isRevNth s1 CID uid PID N using uid eqExcPID-N2-imp[OF
ss1] by auto
      have decs': (reviewsPaper (paper s' PID))!N  $\neq$  [] (reviewsPaper (paper s1'
PID))!N  $\neq$  []
      using nonempty-reviews-persist rs rs1 step step1 uid uid1 decs by blast+
      have ?match proof
        show validTrans ?trn1 using step1 by simp
      next
        show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi 1$  by auto
      next
        show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
      next
        assume  $\gamma$  ?trn thus g ?trn = g ?trn1
        using eqExcPID-N2-step-out[OF ss1 step step1 rsT rs1 uid ph - decs] by
simp
      next
        have  $\Delta 4$  s' vl' s1' vl1 using ph' uuid' s's1' PID' unfolding  $\Delta 4$ -def vl1
vl' by (auto simp: decs')
        thus ? $\Delta$  s' vl' s1' vl1 by simp
      qed
      thus ?thesis by simp
    qed
  qed
  thus ?thesis using vl1 by simp
qed
qed

```

**definition** *K1exit* where

*K1exit* cid s  $\equiv$  PID  $\in \in$  paperIDs s cid  $\wedge$  phase s cid  $>$  revPH  $\wedge$   $\neg$  ( $\exists$  uid. isRevNth s cid uid PID N)

**lemma** *invarNT-K1exit*: invarNT (*K1exit* cid)

**unfolding** *invarNT-def* **apply** (safe dest!: reachNT-reach)

**subgoal** for - a **apply**(cases a)

**subgoal** for x1 **apply**(cases x1) **apply** (fastforce simp add: c-defs *K1exit-def* geq-noPH-confIDs)+ .

**subgoal** for x2 **apply**(cases x2) **apply** (fastforce simp add: u-defs *K1exit-def* paperIDs-equals)+ .

**subgoal** for x3 **apply**(cases x3) **apply** (fastforce simp add: uu-defs *K1exit-def*)+

.

by auto

done

**lemma** *noVal-K1exit*: *noVal (K1exit cid) v*  
**apply**(*rule noφ-noVal*)  
**unfolding** *noφ-def* **apply** *safe*  
**subgoal for** - *a* **apply**(*cases a*)  
  **subgoal by** (*fastforce simp add: c-defs K1exit-def*)  
  **subgoal for** *x2* **apply**(*cases x2*) **apply** (*auto simp add: u-defs K1exit-def*)  
  **apply** (*metis less-not-refl paperIDs-equals reachNT-reach*) .  
  **subgoal for** *x3* **apply**(*cases x3*) **apply** (*auto simp add: uu-defs K1exit-def*)  
  **apply** (*metis isRev-def3 paperIDs-equals reachNT-reach*) .  
  **by** *auto*  
**done**

**definition** *K2exit* **where**  
*K2exit cid s*  $\equiv$  *PID*  $\in$  *paperIDs s cid*  $\wedge$  *phase s cid*  $>$  *disPH*

**lemma** *invarNT-K2exit*: *invarNT (K2exit cid)*  
**unfolding** *invarNT-def* **apply** (*safe dest!: reachNT-reach*)  
**subgoal for** - *a* **apply**(*cases a*)  
  **subgoal for** *x1* **apply**(*cases x1*) **apply** (*fastforce simp add: c-defs K2exit-def*  
*geq-noPH-confIDs*) $+$  .  
  **subgoal for** *x2* **apply**(*cases x2*) **apply** (*fastforce simp add: u-defs K2exit-def*  
*paperIDs-equals*) $+$  .  
  **subgoal for** *x3* **apply**(*cases x3*) **apply** (*fastforce simp add: uu-defs K2exit-def*) $+$   
  .  
  **by** *auto*  
**done**

**lemma** *noVal-K2exit*: *noVal (K2exit cid) v*  
**apply**(*rule noφ-noVal*)  
**unfolding** *noφ-def* **apply** *safe*  
**subgoal for** - *a* **apply**(*cases a*)  
  **subgoal by** (*fastforce simp add: c-defs K2exit-def*)  
  **subgoal for** *x2* **apply**(*cases x2*) **apply** (*auto simp add: u-defs K2exit-def*)  
  **using** *paperIDs-equals reachNT-reach* **apply** *fastforce* .  
  **subgoal for** *x3* **apply**(*cases x3*) **apply** (*auto simp add: uu-defs K2exit-def*)  
  **using** *paperIDs-equals reachNT-reach* **apply** *fastforce* .  
  **by** *auto*  
**done**

**lemma** *unwind-exit-Δe*: *unwind-exit Δe*  
**proof**  
  **fix** *s s1*  $::$  *state* **and** *vl vl1*  $::$  *value list*  
  **assume** *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and**  $\Delta e$ :  $\Delta e s vl s1 vl1$   
  **hence** *vl*: *vl*  $\neq$   $\square$  **using** *reachNT-reach* **unfolding**  $\Delta e$ -*def* **by** *auto*  
  **then obtain** *CID* **where** *K1exit CID s*  $\vee$  *K2exit CID s* **using**  $\Delta e$   
  **unfolding** *K1exit-def K2exit-def Δe-def isREVNth-def* **by** *auto*  
  **thus** *vl*  $\neq$   $\square$   $\wedge$  *exit s (hd vl)* **apply**(*simp add: vl*)  
  **by** (*metis rsT exitI2 invarNT-K1exit noVal-K1exit invarNT-K2exit noVal-K2exit*)  
**qed**

```

theorem secure: secure
apply(rule unwind-decomp4-secure[of  $\Delta 1 \ \Delta 2 \ \Delta e \ \Delta 3 \ \Delta 4$ ])
using
  istate- $\Delta 1$ 
  unwind-cont- $\Delta 1$  unwind-cont- $\Delta 2$  unwind-cont- $\Delta 2$  unwind-cont- $\Delta 3$  unwind-cont- $\Delta 4$ 
  unwind-exit- $\Delta e$ 
by auto

end
theory Review-RAut-NCPC-PAut
imports ../Observation-Setup Review-Value-Setup Bounded-Deducibility-Security.Compositional-Reasoning
begin

```

## 6.5 Confidentiality from users who are not the review's author, a PC member, or an author of the paper

We verify the following property:

A group of users UIDs learn nothing about the various updates to the N'th review of a paper PID (save for the inexistence of any updates) unless/until

- a user in UIDs is the review's author, or
- a user in UIDs becomes a PC member in the paper's conference having no conflict with that paper and the conference moves to the discussion phase, or
- a user in UIDs become a PC member in the paper's conference or an author of the paper and the conference moves to the notification phase

```

type-synonym value = rcontent

```

```

fun f :: (state,act,out) trans  $\Rightarrow$  value where
f (Trans - (Uact (uReview cid uid p pid n rc)) - -) = rc
|
f (Trans - (UUact (uuReview cid uid p pid n rc)) - -) = rc

```

```

fun T :: (state,act,out) trans  $\Rightarrow$  bool where
T (Trans - - ou s') =
  ( $\exists$  uid  $\in$  UIDs.
    isREVNth s' uid PID N
     $\vee$ 
    ( $\exists$  cid. PID  $\in$  paperIDs s' cid  $\wedge$  isPC s' cid uid  $\wedge$  pref s' uid PID  $\neq$  Conflict
     $\wedge$  phase s' cid  $\geq$  disPH)
     $\vee$ 
    ( $\exists$  cid. PID  $\in$  paperIDs s' cid  $\wedge$  isPC s' cid uid  $\wedge$  phase s' cid  $\geq$  notifPH)
  )

```

$$\vee$$

$$isAUT\ s'\ uid\ PID \wedge (\exists\ cid.\ PID \in\in\ paperIDs\ s'\ cid \wedge phase\ s'\ cid \geq\ notifPH)$$

$$)$$

**declare**  $T.simps$  [ $simp\ del$ ]

**definition**  $B :: value\ list \Rightarrow value\ list \Rightarrow bool$  **where**  
 $B\ vl\ vl1 \equiv vl \neq []$

**interpretation**  $BD\text{-}Security\text{-}IO$  **where**  
 $istate = istate$  **and**  $step = step$  **and**  
 $\varphi = \varphi$  **and**  $f = f$  **and**  $\gamma = \gamma$  **and**  $g = g$  **and**  $T = T$  **and**  $B = B$   
**done**

**lemma**  $reachNT\text{-}non\text{-}isPC\text{-}isChair$ :  
**assumes**  $reachNT\ s$  **and**  $uid \in UIDs$   
**shows**

$\neg isRevNth\ s\ cid\ uid\ PID\ N \wedge$   
 $(PID \in\in\ paperIDs\ s\ cid \wedge isPC\ s\ cid\ uid \longrightarrow$   
 $(pref\ s\ uid\ PID = Conflict \vee phase\ s\ cid < disPH) \wedge phase\ s\ cid < notifPH)$   
 $\wedge$   
 $(PID \in\in\ paperIDs\ s\ cid \wedge isChair\ s\ cid\ uid \longrightarrow$   
 $(pref\ s\ uid\ PID = Conflict \vee phase\ s\ cid < disPH) \wedge phase\ s\ cid < notifPH)$   
 $\wedge$   
 $(isAut\ s\ cid\ uid\ PID \longrightarrow phase\ s\ cid < notifPH)$

**using**  $assms$  **apply**  $induct$   
**apply** ( $auto\ simp: istate\text{-}def$ )[]  
**apply** ( $intro\ conjI$ )  
**subgoal** **for**  $trn$  **apply** ( $cases\ trn, simp\ add: T.simps\ reachNT\text{-}reach\ isAUT\text{-}def$   
 $isREVnth\text{-}def$ )[] .  
**subgoal** **for**  $trn$  **apply** ( $cases\ trn, simp\ add: T.simps\ reachNT\text{-}reach\ isAUT\text{-}def$   
 $isREVnth\text{-}def$ )[]  
**apply** ( $metis\ not\text{-}less$ ) .  
**subgoal** **for**  $trn$  **apply** ( $cases\ trn, simp\ add: T.simps\ reachNT\text{-}reach\ isAUT\text{-}def$   
 $isREVnth\text{-}def$ )[]  
**apply** ( $metis\ isChair\text{-}isPC\ not\text{-}less\ reachNT\text{-}reach\ reach\text{-}PairI$ ) .  
**subgoal** **for**  $trn$  **apply** ( $cases\ trn, simp\ add: T.simps\ reachNT\text{-}reach\ isAUT\text{-}def$   
 $isREVnth\text{-}def$ )[]  
**apply** ( $metis\ isAut\text{-}paperIDs\ not\text{-}less\ reachNT\text{-}reach\ reach\text{-}PairI$ ) .  
**done**

**lemma**  $T\text{-}\varphi\text{-}\gamma$ :  
**assumes**  $1: reachNT\ s$  **and**  $2: step\ s\ a = (ou, s')\ \varphi\ (Trans\ s\ a\ ou\ s')$   
**shows**  $\neg\ \gamma\ (Trans\ s\ a\ ou\ s')$   
**using**  $reachNT\text{-}non\text{-}isPC\text{-}isChair[OF\ 1]$   $2$  **unfolding**  $T.simps\ \varphi\text{-}def2$   
**apply** ( $auto\ simp: u\text{-}defs\ uu\text{-}defs\ isRev\text{-}imp\text{-}isRevNth\text{-}getReviewIndex$ )  
**by** ( $metis\ isRev\text{-}imp\text{-}isRevNth\text{-}getReviewIndex$ ) $+$

**lemma**  $eqExcPID\text{-}N\text{-}step\text{-}out$ :

```

assumes  $s's1'$ :  $eqExcPID-N\ s\ s1$ 
and  $step$ :  $step\ s\ a = (ou, s')$  and  $step1$ :  $step\ s1\ a = (ou1, s1')$ 
and  $sT$ :  $reachNT\ s$  and  $s1$ :  $reach\ s1$ 
and  $PID$ :  $PID \in \in paperIDs\ s\ cid$ 
and  $UIDs$ :  $userOfA\ a \in UIDs$ 
shows  $ou = ou1$ 
proof –
  note  $s = reachNT-reach[OF\ sT]$ 
  note  $Inv = reachNT-non-isPC-isChair[OF\ sT\ UIDs]$ 
  note  $eqs = eqExcPID-N-imp[OF\ s's1']$ 
  note  $eqs' = eqExcPID-N-imp1[OF\ s's1']$ 
  note  $eqss = eqExcPID-N-imp2[OF\ s's1']\ eqExcPID-N-imp3'[OF\ s\ s's1']$ 

  note  $simps[simp] = c-defs\ u-defs\ uu-defs\ r-defs\ l-defs\ Paper-dest-conv\ eqExc-$ 
 $cPID-N-def\ eqExcPID-N-def\ eqExcD$ 
  note  $* = step\ step1\ eqs\ eqs'\ eqss\ s\ s1\ UIDs\ PID\ paperIDs-equals[OF\ s]\ Inv$ 

  show  $?thesis$ 
  proof ( $cases\ a$ )
    case ( $Cact\ x1$ )
      with  $*$  show  $?thesis$  by ( $cases\ x1; auto$ )
    next
      case ( $Uact\ x2$ )
        with  $*$  show  $?thesis$  by ( $cases\ x2; auto$ )
      next
        case ( $UUact\ x3$ )
          with  $*$  show  $?thesis$  by ( $cases\ x3; auto$ )
        next
          case ( $Ract\ x4$ )
            with  $*$  show  $?thesis$ 
            proof ( $cases\ x4$ )
              case ( $rMyReview\ x81\ x82\ x83\ x84$ )
                with  $Ract\ *$  show  $?thesis$ 
                by  $clarsimp\ (metis\ getRevRole-Some-Rev-isRevNth)$ 
              next
                case ( $rReviews\ x91\ x92\ x93\ x94$ )
                  with  $Ract\ *$  show  $?thesis$ 
                  by  $clarsimp\ (metis\ not-less)$ 
                next
                  case ( $rFinalReviews\ x121\ x122\ x123\ x124$ )
                    with  $Ract\ *$  show  $?thesis$ 
                    by  $clarsimp\ (metis\ not-less)$ 
                  qed  $auto$ 
                next
                  case ( $Lact\ x5$ )
                    with  $*$  show  $?thesis$  by ( $cases\ x5; auto; presburger$ )
                  qed
                qed
  qed

```

**definition**  $\Delta 1 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 1\ s\ vl\ s1\ vl1 \equiv$   
 $(\forall\ cid.\ PID \in \in paperIDs\ s\ cid \longrightarrow phase\ s\ cid < revPH) \wedge s = s1 \wedge B\ vl\ vl1$

**definition**  $\Delta 2 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 2\ s\ vl\ s1\ vl1 \equiv$   
 $\exists\ cid.$   
 $PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid = revPH \wedge$   
 $\neg (\exists\ uid.\ isREVNth\ s\ uid\ PID\ N) \wedge$   
 $s = s1 \wedge B\ vl\ vl1$

**definition**  $\Delta 3 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 3\ s\ vl\ s1\ vl1 \equiv$   
 $\exists\ cid\ uid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid = revPH \wedge isREVNth\ s\ uid\ PID$   
 $N \wedge eqExcPID-N\ s\ s1$

**definition**  $\Delta 4 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 4\ s\ vl\ s1\ vl1 \equiv$   
 $\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > revPH \wedge eqExcPID-N\ s\ s1 \wedge vl1$   
 $= []$

**definition**  $\Delta e :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta e\ s\ vl\ s1\ vl1 \equiv$   
 $vl \neq [] \wedge$   
 $(\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > revPH \wedge \neg (\exists\ uid.\ isREVNth\ s$   
 $uid\ PID\ N))$

**lemma** *istate- $\Delta 1$* :  
**assumes**  $B: B\ vl\ vl1$   
**shows**  $\Delta 1\ istate\ vl\ istate\ vl1$   
**using**  $B$  **unfolding**  $\Delta 1$ -def  $B$ -def *istate-def* **by** *auto*

**lemma** *unwind-cont- $\Delta 1$* : *unwind-cont*  $\Delta 1\ \{\Delta 1, \Delta 2, \Delta e\}$

**proof**(*rule, simp*)

**let**  $?\Delta = \lambda s\ vl\ s1\ vl1.\ \Delta 1\ s\ vl\ s1\ vl1 \vee \Delta 2\ s\ vl\ s1\ vl1 \vee \Delta e\ s\ vl\ s1\ vl1$

**fix**  $s\ s1 :: state$  **and**  $vl\ vl1 :: value\ list$

**assume**  $rsT: reachNT\ s$  **and**  $rs1: reach\ s1$  **and**  $\Delta 1\ s\ vl\ s1\ vl1$

**hence**  $rs: reach\ s$  **and**  $ss1: s1 = s$  **and**  $vl: vl \neq []$

**and**  $PID$ -ph:  $\bigwedge\ cid.\ PID \in \in paperIDs\ s\ cid \implies phase\ s\ cid < revPH$

**using** *reachNT-reach* **unfolding**  $\Delta 1$ -def  $B$ -def **by** *auto*

**show** *iact*ion  $?\Delta\ s\ vl\ s1\ vl1 \vee$

$((vl = [] \longrightarrow vl1 = []) \wedge reaction\ ?\Delta\ s\ vl\ s1\ vl1)$  (**is**  $?iact \vee (- \wedge ?react)$ )

**proof**–

**have**  $?react$  **proof**

**fix**  $a :: act$  **and**  $ou :: out$  **and**  $s' :: state$  **and**  $vl'$

**let**  $?trn = Trans\ s\ a\ ou\ s'$  **let**  $?trn1 = Trans\ s1\ a\ ou\ s'$

**assume** *step*:  $step\ s\ a = (ou, s')$  **and**  $T: \neg T\ ?trn$  **and**  $c: consume\ ?trn\ vl\ vl'$

```

have  $\varphi: \neg \varphi$  ?trn
  apply(cases a)
  subgoal by simp
    subgoal for x2 apply(cases x2) using step PID-ph by (fastforce simp:
u-defs)+
    subgoal for x3 apply(cases x3) using step PID-ph by (fastforce simp:
uu-defs)+
      by simp-all
      hence  $vl': vl' = vl$  using c unfolding consume-def by auto
      show match  $? \Delta s s1 vl1 a ou s' vl' \vee ignore ? \Delta s s1 vl1 a ou s' vl'$  (is ?match
 $\vee ?ignore$ )
      proof-
        have ?match proof
          show validTrans ?trn1 unfolding ss1 using step by simp
        next
          show consume ?trn1 vl1 vl1 unfolding consume-def ss1 using  $\varphi$  by auto
        next
          show  $\gamma ?trn = \gamma ?trn1$  unfolding ss1 by simp
        next
          assume  $\gamma ?trn$  thus  $g ?trn = g ?trn1$  unfolding ss1 by simp
        next
          show  $? \Delta s' vl' s' vl1$ 
          proof(cases  $\exists cid. PID \in \in paperIDs s cid$ )
            case False note PID = False
            have PID-ph':  $\bigwedge cid. PID \in \in paperIDs s' cid \implies phase s' cid < revPH$ 
using PID step rs
          apply(cases a)
            subgoal for - x1 apply(cases x1) apply(fastforce simp: c-defs)+ .
            subgoal for - x2 apply(cases x2) apply(fastforce simp: u-defs)+ .
            subgoal for - x3 apply(cases x3) apply(fastforce simp: uu-defs)+ .
              by auto
            hence  $\Delta 1 s' vl' s' vl1$  unfolding  $\Delta 1$ -def B-def vl' using PID-ph' vl by
auto
            thus ?thesis by auto
          next
            case True
            then obtain CID where PID:  $PID \in \in paperIDs s CID$  by auto
            hence ph:  $phase s CID < revPH$  using PID-ph by auto
            have PID':  $PID \in \in paperIDs s' CID$  by (metis PID paperIDs-mono
step)
            show ?thesis
            proof(cases  $phase s' CID < revPH$ )
              case True note ph' = True
              hence  $\Delta 1 s' vl' s' vl1$  unfolding  $\Delta 1$ -def B-def vl' using vl ph' PID'
apply auto
              by (metis reach-PairI paperIDs-equals rs step)
              thus ?thesis by auto
            next
              case False note ph' = False

```

```

      have  $\neg (\exists \text{ uid. isRevNth } s \text{ CID uid PID } N)$  using rs ph is-RevNth-geq-revPH by fastforce
      hence ph-isRev':  $\text{phase } s' \text{ CID} = \text{revPH} \wedge \neg (\exists \text{ uid. isRevNth } s' \text{ CID uid PID } N)$ 
      using ph' ph PID step rs
      apply(cases a)
      subgoal for x1 apply(cases x1) apply(fastforce simp: c-defs) $+$  .
      subgoal for x2 apply(cases x2) apply(fastforce simp: u-defs) $+$  .
      subgoal for x3 apply(cases x3) apply(fastforce simp: uu-defs) $+$  .
      by auto
      hence  $\neg (\exists \text{ uid. isREVNth } s' \text{ uid PID } N)$ 
      by (metis PID' isREVNth-imp-isRevNth reach-PairI rs1 ss1 step)
      hence  $\Delta 2 \text{ } s' \text{ } vl' \text{ } s' \text{ } vl1$ 
      unfolding  $\Delta 2\text{-def } B\text{-def isREVNth-def } vl'$  using vl ph' PID' ph-isRev'
by auto
      thus ?thesis by auto
      qed
      qed
      qed
      thus ?thesis by simp
      qed
      qed
      thus ?thesis using vl by auto
      qed
      qed

```

**lemma** *unwind-cont- $\Delta 2$* : *unwind-cont*  $\Delta 2 \{ \Delta 2, \Delta 3, \Delta e \}$

**proof**(*rule, simp*)

**let**  $?\Delta = \lambda s \text{ } vl \text{ } s1 \text{ } vl1. \Delta 2 \text{ } s \text{ } vl \text{ } s1 \text{ } vl1 \vee \Delta 3 \text{ } s \text{ } vl \text{ } s1 \text{ } vl1 \vee \Delta e \text{ } s \text{ } vl \text{ } s1 \text{ } vl1$

**fix** *s s1* :: *state* **and** *vl vl1* :: *value list*

**assume** *rsT*: *reachNT* *s* **and** *rs1*: *reach* *s1* **and**  $\Delta 2 \text{ } s \text{ } vl \text{ } s1 \text{ } vl1$

**then obtain** *CID* **where** *rs*: *reach* *s* **and** *ph*: *phase* *s*  $\text{CID} = \text{revPH}$  (**is** *?ph* = -)

**and** *PID*:  $\text{PID} \in \in \text{paperIDs } s \text{ CID}$  **and** *ss1*:  $s1 = s$

**and** *vl*:  $vl \neq []$  **and** *uid*:  $\neg (\exists \text{ uid. isREVNth } s \text{ uid PID } N)$

**using** *reachNT-reach* **unfolding**  $\Delta 2\text{-def } B\text{-def}$  **by** *auto*

**hence** *uid*:  $\neg (\exists \text{ uid. isRevNth } s \text{ CID uid PID } N)$  **by** (*metis isREVNth-def*)

**show** *iact*ion  $? \Delta \text{ } s \text{ } vl \text{ } s1 \text{ } vl1 \vee$

$((vl = [] \longrightarrow vl1 = [])) \wedge \text{reaction } ? \Delta \text{ } s \text{ } vl \text{ } s1 \text{ } vl1$  (**is** *?iact*  $\vee (- \wedge ?react)$ )

**proof**–

**have** *?react* **proof**

**fix** *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*

**let** *?trn* = *Trans* *s a ou s'* **let** *?trn1* = *Trans* *s1 a ou s'*

**let** *?ph'* = *phase* *s' CID*

**assume** *step*: *step* *s a* = (*ou*, *s'*) **and** *T*:  $\neg T \text{ } ?trn$  **and** *c*: *consume* *?trn vl vl'*

**have**  $\varphi$ :  $\neg \varphi \text{ } ?trn$

**apply**(*cases a*)

**subgoal** **by** *simp*

**subgoal** for *x2* **apply**(*cases x2*)

```

    using step ph uid apply (auto simp: u-defs isRev-def3)
    by (metis PID paperIDs-equals rs)
  subgoal for x3 apply(cases x3)
    using step ph apply (auto simp: uu-defs)
    by (metis PID n-not-Suc-n paperIDs-equals rs)
  by simp-all
  hence vl': vl' = vl using c unfolding consume-def by auto
  have PID': PID ∈∈ paperIDs s' CID by (metis paperIDs-mono step PID)
  show match ?Δ s s1 vl1 a ou s' vl' ∨ ignore ?Δ s s1 vl1 a ou s' vl' (is ?match
  ∨ ?ignore)
  proof-
    have ?match proof
      show validTrans ?trn1 unfolding ss1 using step by simp
    next
      show consume ?trn1 vl1 vl1 unfolding consume-def ss1 using φ by auto
    next
      show γ ?trn = γ ?trn1 unfolding ss1 by simp
    next
      assume γ ?trn thus g ?trn = g ?trn1 unfolding ss1 by simp
    next
      show ?Δ s' vl' s' vl1
      proof(cases ?ph' = revPH)
        case True note ph' = True
        show ?thesis
        proof(cases ∃ uid. isRevNth s' CID uid PID N)
          case False
          hence ¬ (∃ uid. isREVNth s' uid PID N)
          by (metis reach-PairI PID' isREVNth-imp-isRevNth rs1 ss1 step)
          hence Δ2 s' vl' s' vl1 unfolding Δ2-def B-def vl' using vl ph' PID'
        by auto
        thus ?thesis by auto
      next
        case True hence ∃ uid. isREVNth s' uid PID N by (metis is-
        REVNth-def)
        hence Δ3 s' vl' s' vl1 unfolding Δ3-def vl' using vl ph' PID' by auto
        thus ?thesis by auto
      qed
    next
      case False hence 1: ?ph' > revPH ∧ ¬ (∃ uid. isRevNth s' CID uid
      PID N)
      using PID ph uid step rs
      apply(cases a)
      subgoal for x1 apply(cases x1) apply(fastforce simp: c-defs)+ .
      subgoal for x2 apply(cases x2) apply(fastforce simp: u-defs)+ .
      subgoal for x3 apply(cases x3) apply(fastforce simp: uu-defs)+ .
      by auto
      hence ¬ (∃ uid. isREVNth s' uid PID N)
      by (metis reach-PairI PID' isREVNth-imp-isRevNth rs1 ss1 step)
      hence Δe s' vl' s' vl1 unfolding Δe-def vl' using vl PID' 1 by auto

```

```

      thus ?thesis by auto
    qed
  qed
  thus ?thesis by simp
  qed
  qed
  thus ?thesis using vl by simp
  qed
  qed

```

**lemma** *unwind-cont- $\Delta_3$* : *unwind-cont*  $\Delta_3$   $\{\Delta_3, \Delta_4, \Delta_e\}$

**proof**(*rule, simp*)

let  $\Delta = \lambda s vl s1 vl1. \Delta_3 s vl s1 vl1 \vee \Delta_4 s vl s1 vl1 \vee \Delta_e s vl s1 vl1$

**fix**  $s s1 :: state$  **and**  $vl vl1 :: value list$

**assume**  $rsT: reachNT s$  **and**  $rs1: reach s1$  **and**  $\Delta_3 s vl s1 vl1$

**then obtain**  $CID uid$  **where**  $uuid: isREVNth s uid PID N$

**and**  $rs: reach s$  **and**  $ph: phase s CID = revPH (is ?ph = -)$  **and**  $ss1: eqExcPID-N s s1$

**and**  $PID: PID \in \in paperIDs s CID$  **using** *reachNT-reach unfolding  $\Delta_3$ -def* **by** *blast*

**hence**  $uid: isRevNth s CID uid PID N$  **by** (*metis isREVNth-imp-isRevNth*)

**hence**  $uid-notin: uid \notin UIDs$  **using** *reachNT-non-isPC-isChair[OF rsT]* **by** *auto*

**show** *iaction*  $\Delta s vl s1 vl1 \vee$

$((vl = [] \longrightarrow vl1 = []) \wedge reaction \Delta s vl s1 vl1) (is ?iact \vee (- \wedge ?react))$

**proof**(*cases vl1*)

**case** (*Cons v1 vl1'*) **note**  $vl1 = Cons$

**have**  $uid1: isRevNth s CID uid PID N$  **using**  $ss1 uid$  **unfolding** *eqExcPID-N-def*

**by** *auto*

**define**  $a1$  **where**  $a1 \equiv Uact (uReview CID uid (pass s uid) PID N v1)$

**obtain**  $s1' ou1$  **where**  $step1: step s1 a1 = (ou1, s1')$  **by** (*metis prod.exhaust*)

let  $?trn1 = Trans s1 a1 ou1 s1'$

**have**  $s1s1': eqExcPID-N s1 s1'$  **using**  $a1-def step1 uReview-uuReview-step-eqExcPID-N$

**by** *blast*

**have**  $ss1': eqExcPID-N s s1'$  **using** *eqExcPID-N-trans[OF ss1 s1s1']* .

**hence**  $many-s1': PID \in \in paperIDs s1' CID isRevNth s1' CID uid PID N$

$phase s1' CID = revPH pass s1' uid = pass s uid$

**using**  $uid PID ph$  **unfolding** *eqExcPID-N-def* **by** *auto*

**hence**  $more-s1': uid \in \in userIDs s1' CID \in \in confIDs s1'$

**by** (*metis paperIDs-confIDs reach-PairI roles-userIDs rs1 step1 many-s1'(1)*)**+**

**have**  $f: f ?trn1 = v1$  **unfolding**  $a1-def$  **by** *simp*

**have**  $rs1': reach s1'$  **using**  $rs1 step1$  **by** (*auto intro: reach-PairI*)

**have**  $ou1: ou1 = outOK$

**using**  $step1 uid1 ph$  **unfolding**  $a1-def$  **apply** (*simp add: u-defs uu-defs*

*many-s1' more-s1' isRev-def2*)

**by** (*metis isRevNth-getReviewIndex many-s1' rs1*)

**have**  $?iact$  **proof**

**show**  $step s1 a1 = (ou1, s1')$  **by** *fact*

**next**

**show**  $\varphi: \varphi ?trn1$  **using**  $ou1$  **unfolding**  $a1-def$  **by** *simp*

```

    thus consume ?trn1 vl1 vl1' using f unfolding consume-def vl1 by simp
  next
  show  $\neg \gamma$  ?trn1 by (simp add: a1-def uid-notin)
  next
  have  $\Delta 3$  s vl s1' vl1' unfolding  $\Delta 3$ -def using ph PID ss1' uuid by auto
  thus  $?\Delta$  s vl s1' vl1' by simp
  qed
  thus ?thesis by auto
  next
  case Nil note vl1 = Nil
  have ?react proof
    fix a :: act and ou :: out and s' :: state and vl'
    let ?trn = Trans s a ou s'
    let ?ph' = phase s' CID
    assume step: step s a = (ou, s') and T:  $\neg T$  ?trn and c: consume ?trn vl vl'
    have PID': PID  $\in$  paperIDs s' CID using PID rs by (metis paperIDs-mono
  step)
    have uid': isRevNth s' CID uid PID N
    using uid step rs ph PID isRevNth-persistent by auto
    have uuid': isREVNth s' uid PID N by (metis isREVNth-def uid')
    show match  $?\Delta$  s s1 vl1 a ou s' vl'  $\vee$  ignore  $?\Delta$  s s1 vl1 a ou s' vl' (is ?match
 $\vee$  ?ignore)
    proof(cases  $\varphi$  ?trn)
      case False note  $\varphi = \text{False}$ 
      have vl: vl' = vl using c  $\varphi$  unfolding consume-def by (cases vl) auto
      obtain ou1 and s1' where step1: step s1 a = (ou1, s1') by (metis
  prod.exhaust)
      let ?trn1 = Trans s1 a ou1 s1'
      have s's1': eqExcPID-N s' s1' using eqExcPID-N-step[OF ss1 step step1] .
      have  $\varphi 1$ :  $\neg \varphi$  ?trn1 using  $\varphi$  unfolding eqExcPID-N-step- $\varphi$ [OF ss1 step
  step1] .
      have ?match proof
        show validTrans ?trn1 using step1 by simp
      next
      show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi 1$  by auto
      next
      show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
      next
      assume  $\gamma$  ?trn thus g ?trn = g ?trn1
      using eqExcPID-N-step-out[OF ss1 step step1 rsT rs1 PID] by simp
      next
      show  $?\Delta$  s' vl' s1' vl1
      proof(cases ?ph' = revPH)
        case True
          hence  $\Delta 3$  s' vl' s1' vl1 using PID' s's1' uuid' unfolding  $\Delta 3$ -def by
  auto
        thus ?thesis by auto
      next
        case False hence ?ph' > revPH

```

```

      using ph rs step by (metis le-less phase-increases2 prod.sel(2))
      hence  $\Delta_4 s' vl' s1' vl1$  using s's1' PID' unfolding  $\Delta_4$ -def vl1 by auto
      thus ?thesis by auto
    qed
  qed
  thus ?thesis by simp
next
  case True note  $\varphi = \text{True}$ 
  have s's: eqExcPID-N s' s using eqExcPID-N-sym[OF  $\varphi$ -step-eqExcPID-N[OF
 $\varphi$  step]] .
  have s's1: eqExcPID-N s' s1 using eqExcPID-N-trans[OF s's ss1] .
  have ?ignore proof
    show  $\neg \gamma$  ?trn using T- $\varphi$ - $\gamma$   $\varphi$  rsT step by auto
  next
    show  $? \Delta s' vl' s1 vl1$ 
    proof(cases ?ph' = revPH)
      case True
      hence  $\Delta_3 s' vl' s1 vl1$  using s's1 PID' uuid' unfolding  $\Delta_3$ -def by auto
      thus ?thesis by auto
    next
      case False hence  $?ph' > \text{revPH}$ 
      using ph rs step using eqExcPID-N-def s's by auto
      hence  $\Delta_4 s' vl' s1 vl1$  using s's1 PID' unfolding  $\Delta_4$ -def vl1 by auto
      thus ?thesis by auto
    qed
  qed
  thus ?thesis by auto
  qed
  thus ?thesis using vl1 by auto
  qed
  qed

```

```

lemma unwind-cont- $\Delta_4$ : unwind-cont  $\Delta_4$  { $\Delta_4, \Delta e$ }
proof(rule, simp)
  let  $? \Delta = \lambda s vl s1 vl1. \Delta_4 s vl s1 vl1 \vee \Delta e s vl s1 vl1$ 
  fix s s1 :: state and vl vl1 :: value list
  assume rsT: reachNT s and rs1: reach s1 and  $\Delta_4 s vl s1 vl1$ 
  then obtain CID where rs: reach s and ph: phase s CID > revPH (is ?ph >
  -)
  and PID: PID  $\in \in$  paperIDs s CID and ss1: eqExcPID-N s s1 and vl1: vl1 = []
  using reachNT-reach unfolding  $\Delta_4$ -def by auto
  show iaction  $? \Delta s vl s1 vl1 \vee$ 
    ( $(vl = [] \longrightarrow vl1 = []) \wedge$  reaction ? $\Delta$  s vl s1 vl1) (is ?iact  $\vee$  ( $- \wedge ?react$ ))
  proof-
    have ?react
    proof
      fix a :: act and ou :: out and s' :: state and vl'
      let ?trn = Trans s a ou s'

```

```

    let ?ph' = phase s' CID
    assume step: step s a = (ou, s') and T:  $\neg T$  ?trn and c: consume ?trn vl vl'
    have PID': PID  $\in$  paperIDs s' CID using PID rs by (metis paperIDs-mono
step)
    have ph': phase s' CID > revPH using ph rs by (meson less-le-trans local.step
phase-increases)
    show match ? $\Delta$  s s1 vl1 a ou s' vl'  $\vee$  ignore ? $\Delta$  s s1 vl1 a ou s' vl' (is ?match
 $\vee$  ?ignore)
    proof(cases  $\varphi$  ?trn)
      case False note  $\varphi = \text{False}$ 
      have vl: vl' = vl using c  $\varphi$  unfolding consume-def by (cases vl) auto
      obtain ou1 and s1' where step1: step s1 a = (ou1, s1') by (metis
prod.exhaust)
      let ?trn1 = Trans s1 a ou1 s1'
      have s's1': eqExcPID-N s' s1' using eqExcPID-N-step[OF ss1 step step1] .
      have  $\varphi$ 1:  $\neg \varphi$  ?trn1 using  $\varphi$  unfolding eqExcPID-N-step- $\varphi$ [OF ss1 step
step1] .
      have ?match proof
        show validTrans ?trn1 using step1 by simp
      next
        show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi$ 1 by auto
      next
        show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
      next
        assume  $\gamma$  ?trn thus g ?trn = g ?trn1
        using eqExcPID-N-step-out[OF ss1 step step1 rsT rs1 PID] by simp
      next
        have  $\Delta$ 4 s' vl' s1' vl1 using ph' PID' s's1' unfolding  $\Delta$ 4-def vl1 by
auto
        thus ? $\Delta$  s' vl' s1' vl1 by simp
      qed
    thus ?thesis by simp
  next
  case True note  $\varphi = \text{True}$ 
  have s's: eqExcPID-N s' s using eqExcPID-N-sym[OF  $\varphi$ -step-eqExcPID-N[OF
 $\varphi$  step]] .
  have s's1: eqExcPID-N s' s1 using eqExcPID-N-trans[OF s's ss1] .
  have ?ignore proof
    show  $\neg \gamma$  ?trn using T- $\varphi$ - $\gamma$   $\varphi$  rsT step by auto
  next
    have  $\Delta$ 4 s' vl' s1 vl1 using s's1 PID' ph' vl1 unfolding  $\Delta$ 4-def by auto
    thus ? $\Delta$  s' vl' s1 vl1 by auto
  qed
  thus ?thesis by simp
qed
qed
thus ?thesis using vl1 by simp
qed
qed

```

**definition** *K1exit* **where**

*K1exit cid s*  $\equiv$  *PID*  $\in\in$  *paperIDs s cid*  $\wedge$  *phase s cid*  $>$  *revPH*  $\wedge$   $\neg$  ( $\exists$  *uid. isRevNth s cid uid PID N*)

**lemma** *invarNT-K1exit*: *invarNT (K1exit cid)*

**unfolding** *invarNT-def* **apply** (*safe dest!*: *reachNT-reach*)

**subgoal for** - *a* **apply**(*cases a*)

**subgoal for** *x1* **apply**(*cases x1*) **apply** (*fastforce simp add: c-defs K1exit-def geq-noPH-confIDs*) $+$  .

**subgoal for** *x2* **apply**(*cases x2*) **apply** (*fastforce simp add: u-defs K1exit-def paperIDs-equals*) $+$  .

**subgoal for** *x3* **apply**(*cases x3*) **apply** (*fastforce simp add: uu-defs K1exit-def*) $+$

.

**by** *auto*

**done**

**lemma** *noVal-K1exit*: *noVal (K1exit cid) v*

**apply**(*rule no $\varphi$ -noVal*)

**unfolding** *no $\varphi$ -def* **apply** *safe*

**subgoal for** - *a* **apply**(*cases a*)

**subgoal by** (*fastforce simp add: c-defs K1exit-def*)

**subgoal for** *x2* **apply**(*cases x2*) **apply** (*auto simp add: u-defs K1exit-def*)

**apply** (*metis less-not-refl paperIDs-equals reachNT-reach*) .

**subgoal for** *x3* **apply**(*cases x3*) **apply** (*auto simp add: uu-defs K1exit-def*)

**apply** (*metis isRev-def3 paperIDs-equals reachNT-reach*) .

**by** *auto*

**done**

**lemma** *unwind-exit- $\Delta e$* : *unwind-exit  $\Delta e$*

**proof**

**fix** *s s1*  $::$  *state* **and** *vl vl1*  $::$  *value list*

**assume** *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and**  $\Delta e$ :  $\Delta e s vl s1 vl1$

**hence** *vl*: *vl*  $\neq$  [] **using** *reachNT-reach* **unfolding**  $\Delta e$ -*def* **by** *auto*

**then obtain** *CID* **where** *K1exit CID s* **using**  $\Delta e$  **unfolding** *K1exit-def*  $\Delta e$ -*def* *isREVnth-def* **by** *auto*

**thus** *vl*  $\neq$  []  $\wedge$  *exit s (hd vl)* **apply**(*simp add: vl*)

**by** (*metis rsT exitI2 invarNT-K1exit noVal-K1exit*)

**qed**

**theorem** *secure*: *secure*

**apply**(*rule unwind-decomp4-secure*[*of*  $\Delta 1$   $\Delta 2$   $\Delta e$   $\Delta 3$   $\Delta 4$ ])

**using**

*istate- $\Delta 1$*

*unwind-cont- $\Delta 1$*  *unwind-cont- $\Delta 2$*  *unwind-cont- $\Delta 3$*  *unwind-cont- $\Delta 4$*

*unwind-exit- $\Delta e$*

**by** *auto*

```

end
theory Review-All
imports
  Review-RAut
  Review-RAut-NCPC
  Review-RAut-NCPC-PAut
begin

```

```

end
theory Discussion-Intro
imports ../Safety-Properties
begin

```

## 7 Discussion Confidentiality

In this section, we prove confidentiality for the discussion log (with comments made by PC members) on submitted papers. The secrets (values) of interest are therefore the different updates of (i.e., comments posted as part of) the discussion of a given paper with id PID.

Here, we have only one point of compromise between the bound and the trigger (which yields one property): the trigger being “PC membership having no conflict with that paper and the conference having moved to the discussion stage” and the bound allowing to learn almost nothing.

```

end
theory Discussion-Value-Setup
imports Discussion-Intro
begin

```

The ID of the paper under scrutiny:

```

consts PID :: paperID

```

### 7.1 Preliminaries

```

declare updates-commute-paper[simp]

```

```

fun eqExcD :: paper  $\Rightarrow$  paper  $\Rightarrow$  bool where
eqExcD (Paper title abstract ct reviews dis decs)
  (Paper title1 abstract1 ct1 reviews1 dis1 decs1) =
  (title = title1  $\wedge$  abstract = abstract1  $\wedge$  ct = ct1  $\wedge$  reviews = reviews1  $\wedge$  decs =
  decs1)

```

```

lemma eqExcD:
eqExcD pap pap1 =
  (titlePaper pap = titlePaper pap1  $\wedge$  abstractPaper pap = abstractPaper pap1  $\wedge$ 
  contentPaper pap = contentPaper pap1  $\wedge$ 

```

*reviewsPaper pap = reviewsPaper pap1*  $\wedge$  *decsPaper pap = decsPaper pap1*)  
**by**(*cases pap, cases pap1, auto*)

**lemma** *eqExcD-eq[simp,intro!]*: *eqExcD pap pap*  
**by**(*cases pap*) *auto*

**lemma** *eqExcD-sym*:  
**assumes** *eqExcD pap pap1*  
**shows** *eqExcD pap1 pap*  
**apply**(*cases pap, cases pap1*)  
**using** *assms* **by** *auto*

**lemma** *eqExcD-trans*:  
**assumes** *eqExcD pap pap1* **and** *eqExcD pap1 pap2*  
**shows** *eqExcD pap pap2*  
**apply**(*cases pap, cases pap1, cases pap2*)  
**using** *assms* **by** *auto*

**definition** *eeqExcPID* **where**  
*eeqExcPID paps paps1*  $\equiv$   
 $\forall$  *pid*. *if pid = PID then eqExcD (paps pid) (paps1 pid) else paps pid = paps1 pid*

**lemma** *eeqExcPID-eeq[simp,intro!]*: *eeqExcPID s s*  
**unfolding** *eeqExcPID-def* **by** *auto*

**lemma** *eeqExcPID-sym*:  
**assumes** *eeqExcPID s s1* **shows** *eeqExcPID s1 s*  
**using** *assms eqExcD-sym* **unfolding** *eeqExcPID-def* **by** *auto*

**lemma** *eeqExcPID-trans*:  
**assumes** *eeqExcPID s s1* **and** *eeqExcPID s1 s2* **shows** *eeqExcPID s s2*  
**using** *assms eqExcD-trans* **unfolding** *eeqExcPID-def* **by** *simp blast*

**lemma** *eeqExcPID-imp*:  
*eeqExcPID paps paps1*  $\implies$  *eqExcD (paps PID) (paps1 PID)*  
 $\llbracket$ *eeqExcPID paps paps1; pid  $\neq$  PID* $\rrbracket \implies$  *paps pid = paps1 pid*  
**unfolding** *eeqExcPID-def* **by** *auto*

**lemma** *eeqExcPID-cong*:  
**assumes** *eeqExcPID paps paps1*  
**and** *pid = PID*  $\implies$  *eqExcD uu uu1*  
**and** *pid  $\neq$  PID*  $\implies$  *uu = uu1*  
**shows** *eeqExcPID (paps (pid := uu)) (paps1 (pid := uu1))*  
**using** *assms* **unfolding** *eeqExcPID-def* **by** *auto*

**lemma** *eeqExcPID-RDD*:  
*eeqExcPID paps paps1*  $\implies$   
*titlePaper (paps PID) = titlePaper (paps1 PID)  $\wedge$*

$abstractPaper (paps PID) = abstractPaper (paps1 PID) \wedge$   
 $contentPaper (paps PID) = contentPaper (paps1 PID) \wedge$   
 $reviewsPaper (paps PID) = reviewsPaper (paps1 PID) \wedge$   
 $decsPaper (paps PID) = decsPaper (paps1 PID)$   
**using** *eeqExcPID-def* **unfolding** *eqExcD* **by** *auto*

**definition** *eqExcPID* :: *state*  $\Rightarrow$  *state*  $\Rightarrow$  *bool* **where**

$eqExcPID s s1 \equiv$   
 $confIDs s = confIDs s1 \wedge conf s = conf s1 \wedge$   
 $userIDs s = userIDs s1 \wedge pass s = pass s1 \wedge user s = user s1 \wedge roles s = roles$   
 $s1 \wedge$   
 $paperIDs s = paperIDs s1$   
 $\wedge$   
 $eeqExcPID (paper s) (paper s1)$   
 $\wedge$   
 $pref s = pref s1 \wedge$   
 $voronkov s = voronkov s1 \wedge$   
 $news s = news s1 \wedge phase s = phase s1$

**lemma** *eqExcPID-eq[simp,intro!]*: *eqExcPID* *s s*  
**unfolding** *eqExcPID-def* **by** *auto*

**lemma** *eqExcPID-sym*:

**assumes** *eqExcPID* *s s1* **shows** *eqExcPID* *s1 s*  
**using** *assms* *eqExcPID-sym* **unfolding** *eqExcPID-def* **by** *auto*

**lemma** *eqExcPID-trans*:

**assumes** *eqExcPID* *s s1* **and** *eqExcPID* *s1 s2* **shows** *eqExcPID* *s s2*  
**using** *assms* *eqExcPID-trans* **unfolding** *eqExcPID-def* **by** *auto*

**lemma** *eqExcPID-imp*:

$eqExcPID s s1 \implies$   
 $confIDs s = confIDs s1 \wedge conf s = conf s1 \wedge$   
 $userIDs s = userIDs s1 \wedge pass s = pass s1 \wedge user s = user s1 \wedge roles s = roles$   
 $s1 \wedge$   
 $paperIDs s = paperIDs s1$   
 $\wedge$   
 $eeqExcPID (paper s) (paper s1)$   
 $\wedge$   
 $pref s = pref s1 \wedge$   
 $voronkov s = voronkov s1 \wedge$   
 $news s = news s1 \wedge phase s = phase s1 \wedge$

$getAllPaperIDs s = getAllPaperIDs s1 \wedge$   
 $isRev s cid uid pid = isRev s1 cid uid pid \wedge$   
 $getReviewIndex s cid uid pid = getReviewIndex s1 cid uid pid \wedge$   
 $getRevRole s cid uid pid = getRevRole s1 cid uid pid$

**unfolding** *eqExcPID-def getAllPaperIDs-def*  
**unfolding** *isRev-def getReviewIndex-def getRevRole-def* **by** *auto*

**lemma** *eqExcPID-imp1*:  
 $eqExcPID\ s\ s1 \implies eqExcD\ (paper\ s\ pid)\ (paper\ s1\ pid)$   
 $eqExcPID\ s\ s1 \implies pid \neq PID \vee PID \neq pid \implies$   
 $paper\ s\ pid = paper\ s1\ pid \wedge$   
 $getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$   
**unfolding** *eqExcPID-def getNthReview-def eqExcPID-def*  
**apply** *auto*  
**by** (*metis eqExcD-eq*)

**lemma** *eqExcPID-imp2*:  
**assumes** *eqExcPID s s1* **and**  $pid \neq PID \vee PID \neq pid$   
**shows**  $getReviewersReviews\ s\ cid\ pid = getReviewersReviews\ s1\ cid\ pid$   
**proof** –  
**have**  
 $(\lambda uID. if\ isRev\ s\ cid\ uID\ pid\ then\ [(uID,\ getNthReview\ s\ pid\ (getReviewIndex\ s\ cid\ uID\ pid))] \ else\ []) =$   
 $(\lambda uID. if\ isRev\ s1\ cid\ uID\ pid\ then\ [(uID,\ getNthReview\ s1\ pid\ (getReviewIndex\ s1\ cid\ uID\ pid))] \ else\ [])$   
**apply**(*rule ext*)  
**using** *assms by (auto simp: eqExcPID-imp eqExcPID-imp1)*  
**thus** *?thesis unfolding getReviewersReviews-def using assms by (simp add: eqExcPID-imp)*  
**qed**

**lemma** *eqExcPID-RDD*:  
 $eqExcPID\ s\ s1 \implies$   
 $titlePaper\ (paper\ s\ PID) = titlePaper\ (paper\ s1\ PID) \wedge$   
 $abstractPaper\ (paper\ s\ PID) = abstractPaper\ (paper\ s1\ PID) \wedge$   
 $contentPaper\ (paper\ s\ PID) = contentPaper\ (paper\ s1\ PID) \wedge$   
 $reviewsPaper\ (paper\ s\ PID) = reviewsPaper\ (paper\ s1\ PID) \wedge$   
 $decsPaper\ (paper\ s\ PID) = decsPaper\ (paper\ s1\ PID)$   
**using** *eqExcPID-imp eqExcPID-RDD by auto*

**lemma** *eqExcPID-cong[simp, intro]*:  
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|confIDs := uu1|))\ (s1\ (\!|confIDs := uu2|))$   
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|conf := uu1|))\ (s1\ (\!|conf := uu2|))$   
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|userIDs := uu1|))\ (s1\ (\!|userIDs := uu2|))$   
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|pass := uu1|))\ (s1\ (\!|pass := uu2|))$   
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|user := uu1|))\ (s1\ (\!|user := uu2|))$   
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|roles := uu1|))\ (s1\ (\!|roles := uu2|))$

$\langle roles := uu2 \rangle$

$\bigwedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s \langle paperIDs := uu1 \rangle)$   
 $(s1 \langle paperIDs := uu2 \rangle)$   
 $\bigwedge uu1 uu2. eqExcPID s s1 \implies eqExcPID uu1 uu2 \implies eqExcPID (s \langle paper :=$   
 $uu1 \rangle) (s1 \langle paper := uu2 \rangle)$

$\bigwedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s \langle pref := uu1 \rangle) (s1$   
 $\langle pref := uu2 \rangle)$   
 $\bigwedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s \langle voronkov := uu1 \rangle)$   
 $(s1 \langle voronkov := uu2 \rangle)$   
 $\bigwedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s \langle news := uu1 \rangle) (s1$   
 $\langle news := uu2 \rangle)$   
 $\bigwedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s \langle phase := uu1 \rangle) (s1$   
 $\langle phase := uu2 \rangle)$

**unfolding** *eqExcPID-def by auto*

**lemma** *eqExcPID-Paper:*

**assumes**  $s's1'$ : *eqExcPID s s1*

**and** *paper s pid = Paper title abstract content reviews dis decs*

**and** *paper s1 pid = Paper title1 abstract1 content1 reviews1 dis1 decs1*

**shows**  $title = title1 \wedge abstract = abstract1 \wedge content = content1 \wedge reviews =$   
 $reviews1 \wedge decs = decs1$

**using** *assms unfolding eqExcPID-def apply (auto simp: eqExcD eqExcPID-def)*

**by** (*metis titlePaper.simps abstractPaper.simps contentPaper.simps reviewsPaper.simps*  
*decsPaper.simps*  
)+

## 7.2 Value Setup

**type-synonym** *value = string*

**fun**  $\varphi :: (state,act,out) trans \implies bool$  **where**

$\varphi (Trans - (UUact (uuDis cid uid p pid com)) ou -) = (pid = PID \wedge ou = outOK)$

|

$\varphi - = False$

**lemma**  $\varphi$ -def2:

$\varphi (Trans s a ou s') = (\exists cid uid p com. a = UUact (uuDis cid uid p PID com) \wedge$   
 $ou = outOK)$

**proof** (*cases a*)

**case** (*UUact x3*)

**then show** *?thesis by (cases x3; auto)*

**qed** *auto*

**fun**  $f :: (state,act,out) trans \implies value$  **where**

$f (Trans - (UUact (uuDis cid uid p pid com)) -) = com$

**lemma** *UUact-uuDis-step-eqExcPID:*

**assumes**  $a: a = UUact (uuDis\ cid\ uid\ p\ PID\ com)$   
**and**  $step\ s\ a = (ou, s')$   
**shows**  $eqExcPID\ s\ s'$   
**using** *assms* **unfolding**  $eqExcPID-def\ eqExcPID-def$  **by**  $(auto\ simp: uu-defs)$

**lemma**  $\varphi$ -step- $eqExcPID$ :  
**assumes**  $\varphi: \varphi (Trans\ s\ a\ ou\ s')$   
**and**  $s: step\ s\ a = (ou, s')$   
**shows**  $eqExcPID\ s\ s'$   
**using**  $\varphi\ UUact-uuDis-step-eqExcPID[OF\ -\ s]$  **unfolding**  $\varphi-def2$  **by** *blast*

**lemma**  $eqExcPID$ -step:  
**assumes**  $s's1'$ :  $eqExcPID\ s\ s1$   
**and**  $step$ :  $step\ s\ a = (ou, s')$   
**and**  $step1$ :  $step\ s1\ a = (ou1, s1')$   
**shows**  $eqExcPID\ s'\ s1'$   
**proof** –  
**note**  $eqs = eqExcPID-imp[OF\ s's1']$   
**note**  $eqs' = eqExcPID-imp1[OF\ s's1']$   
**note**  $simps[simp] = c-defs\ u-defs\ uu-defs\ r-defs\ l-defs\ Paper-dest-conv\ eqExcPID-def\ eqExcPID-def\ eqExcD$   
**note**  $* = step\ step1\ eqs\ eqs'$

**then show** *?thesis*  
**proof** (*cases a*)  
**case** (*Cact x1*)  
**then show** *?thesis* **using**  $*$  **by** (*cases x1; auto*)  
**next**  
**case** (*Uact x2*)  
**then show** *?thesis* **using**  $*$  **by** (*cases x2; auto*)  
**next**  
**case** (*UUact x3*)  
**then show** *?thesis* **using**  $*$  **by** (*cases x3; auto*)  
**qed** *auto*  
**qed**

**lemma**  $eqExcPID$ -step- $\varphi$ -imp:  
**assumes**  $s's1'$ :  $eqExcPID\ s\ s1$   
**and**  $step$ :  $step\ s\ a = (ou, s')$  **and**  $step1$ :  $step\ s1\ a = (ou1, s1')$   
**and**  $\varphi: \varphi (Trans\ s\ a\ ou\ s')$   
**shows**  $\varphi (Trans\ s1\ a\ ou1\ s1')$   
**using** *assms* **unfolding**  $\varphi-def2$  **by**  $(auto\ simp\ add: uu-defs\ eqExcPID-imp)$

**lemma**  $eqExcPID$ -step- $\varphi$ :  
**assumes**  $s's1'$ :  $eqExcPID\ s\ s1$   
**and**  $step$ :  $step\ s\ a = (ou, s')$  **and**  $step1$ :  $step\ s1\ a = (ou1, s1')$   
**shows**  $\varphi (Trans\ s\ a\ ou\ s') = \varphi (Trans\ s1\ a\ ou1\ s1')$   
**by** (*metis eqExcPID-step- $\varphi$ -imp eqExcPID-sym assms*)

```

end
theory Discussion-NCPC
imports ../Observation-Setup Discussion-Value-Setup Bounded-Deducibility-Security.Compositional-Reasoning
begin

```

### 7.3 Confidentiality protection from non-PC-members

We verify the following property:

A group of users UIDs learn nothing about the various updates of a paper's discussion (i.e., about the comments being posted on a paper by the PC members) (save for the non-existence of any edit) unless/until a user in UIDs becomes a PC member in the paper's conference having no conflict with that paper and the conference moves to the discussion phase.

```

fun T :: (state,act,out) trans  $\Rightarrow$  bool where
  T (Trans - - ou s') =
    ( $\exists$  uid  $\in$  UIDs.  $\exists$  cid.
      PID  $\in$  paperIDs s' cid  $\wedge$  isPC s' cid uid  $\wedge$  pref s' uid PID  $\neq$  Conflict  $\wedge$  phase
      s' cid  $\geq$  disPH
    )

```

```

declare T.simps [simp del]

```

```

definition B :: value list  $\Rightarrow$  value list  $\Rightarrow$  bool where
  B vl vl1  $\equiv$  vl  $\neq$  []

```

```

interpretation BD-Security-IO where
  istate = istate and step = step and
   $\varphi = \varphi$  and  $f = f$  and  $\gamma = \gamma$  and  $g = g$  and  $T = T$  and  $B = B$ 
done

```

```

lemma reachNT-non-isPC-isChair:
assumes reachNT s and uid  $\in$  UIDs
shows
  (PID  $\in$  paperIDs s cid  $\wedge$  isPC s cid uid  $\wedge$  phase s cid  $\geq$  disPH  $\longrightarrow$  pref s uid
  PID = Conflict)  $\wedge$ 
  (PID  $\in$  paperIDs s cid  $\wedge$  isChair s cid uid  $\wedge$  phase s cid  $\geq$  disPH  $\longrightarrow$  pref s
  uid PID = Conflict)
  using assms
  apply induct
  apply (auto simp: istate-def)[]
  apply (intro conjI)
  subgoal for trn apply (cases trn, auto simp: T.simps reachNT-reach)[] .
  by (metis T.elims(3) isChair-isPC reachNT-reach reach.Step tgtOf-simps)

```

```

lemma  $T\text{-}\varphi\text{-}\gamma$ :
assumes 1:  $\text{reachNT } s$  and 2:  $\text{step } s \ a = (ou, s') \ \varphi \ (\text{Trans } s \ a \ ou \ s')$ 
shows  $\neg \gamma \ (\text{Trans } s \ a \ ou \ s')$ 
using  $\text{reachNT-non-isPC-isChair}[OF \ 1] \ 2$  unfolding  $\varphi\text{-def2}$ 
by ( $\text{fastforce simp add: uu-defs}$ )

lemma  $\text{eqExcPID-step-out}$ :
assumes  $s's1'$ :  $\text{eqExcPID } s \ s1$ 
and  $\text{step}$ :  $\text{step } s \ a = (ou, s')$  and  $\text{step1}$ :  $\text{step } s1 \ a = (ou1, s1')$ 
and  $sT$ :  $\text{reachNT } s$  and  $s1$ :  $\text{reach } s1$ 
and  $PID$ :  $PID \in \in \text{paperIDs } s \ cid$ 
and  $UIDs$ :  $\text{userOfA } a \in \text{UIDs}$ 
shows  $ou = ou1$ 
proof -
  note  $Inv = \text{reachNT-non-isPC-isChair}[OF \ sT \ \text{UIDs}]$ 
  note  $\text{eqs} = \text{eqExcPID-imp}[OF \ s's1']$ 
  note  $\text{eqs}' = \text{eqExcPID-imp1}[OF \ s's1']$ 
  note  $s = \text{reachNT-reach}[OF \ sT]$ 

  note  $\text{simps}[simp] = c\text{-defs } u\text{-defs } uu\text{-defs } r\text{-defs } l\text{-defs } \text{Paper-dest-conv } \text{eqExcPID-def } \text{eqExcPID-def } \text{eqExcD}$ 
  note  $*$  =  $\text{step } \text{step1 } \text{eqs } \text{eqs}' \ s \ s1 \ \text{PID } \text{UIDs } \text{paperIDs-equals}[OF \ s] \ \text{Inv}$ 

show  $?thesis$ 
proof ( $\text{cases } a$ )
  case ( $\text{Cact } x1$ )
    then show  $?thesis$  using  $*$  by ( $\text{cases } x1$ )  $\text{auto}$ 
  next
    case ( $\text{Uact } x2$ )
      then show  $?thesis$  using  $*$  by ( $\text{cases } x2$ )  $\text{auto}$ 
    next
      case ( $\text{UUact } x3$ )
        then show  $?thesis$  using  $*$  by ( $\text{cases } x3$ )  $\text{auto}$ 
    next
      case ( $\text{Ract } x4$ )
        show  $?thesis$ 
        proof ( $\text{cases } x4$ )
          case ( $\text{rMyReview } x81 \ x82 \ x83 \ x84$ )
            then show  $?thesis$  using  $*$   $\text{Ract}$  by ( $\text{auto simp add: getNthReview-def}$ )
          next
            case ( $\text{rReviews } x91 \ x92 \ x93 \ x94$ )
              then show  $?thesis$  using  $*$   $\text{Ract}$  by ( $\text{clarsimp;metis eqExcPID-imp2 } s's1'$ )
            next
              case ( $\text{rDis } x111 \ x112 \ x113 \ x114$ )
                then show  $?thesis$  using  $*$   $\text{Ract}$  by ( $\text{clarsimp;metis discussion.inject}$ )
              qed ( $\text{use } * \ \text{Ract in auto}$ )
            next

```

**case** (*Lact x5*)  
**then show** *?thesis* **using** \* **by** (*cases x5; auto; presburger*)  
**qed**  
**qed**

**definition**  $\Delta 1 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 1\ s\ vl\ s1\ vl1 \equiv$   
 $(\forall\ cid.\ PID \in \in paperIDs\ s\ cid \longrightarrow phase\ s\ cid < disPH) \wedge s = s1 \wedge B\ vl\ vl1$

**definition**  $\Delta 2 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 2\ s\ vl\ s1\ vl1 \equiv$   
 $\exists\ cid\ uid.$   
 $PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid = disPH \wedge$   
 $isPC\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict$   
 $\wedge eqExcPID\ s\ s1$

**definition**  $\Delta 3 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 3\ s\ vl\ s1\ vl1 \equiv$   
 $\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > disPH \wedge eqExcPID\ s\ s1 \wedge vl1 =$   
 $\square$

**definition**  $\Delta e :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta e\ s\ vl\ s1\ vl1 \equiv$   
 $vl \neq \square \wedge$   
 $(\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid \geq disPH \wedge$   
 $\neg (\exists\ uid.\ isPC\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict)$   
 $)$

**lemma** *init- $\Delta 1$* :  
**assumes** *B*:  $B\ vl\ vl1$   
**shows**  $\Delta 1\ istate\ vl\ istate\ vl1$   
**using** *B* **unfolding**  $\Delta 1$ -def *B*-def *istate*-def **by** *auto*

**lemma** *unwind-cont- $\Delta 1$* : *unwind-cont*  $\Delta 1\ \{\Delta 1, \Delta 2, \Delta e\}$

**proof**(*rule, simp*)

**let**  $?\Delta = \lambda s\ vl\ s1\ vl1.\ \Delta 1\ s\ vl\ s1\ vl1 \vee \Delta 2\ s\ vl\ s1\ vl1 \vee \Delta e\ s\ vl\ s1\ vl1$

**fix**  $s\ s1 :: state$  **and**  $vl\ vl1 :: value\ list$

**assume**  $rsT$ : *reachNT*  $s$  **and**  $rs1$ : *reach*  $s1$  **and**  $\Delta 1\ s\ vl\ s1\ vl1$

**hence**  $rs$ : *reach*  $s$  **and**  $ss1$ :  $s1 = s$  **and**  $vl$ :  $vl \neq \square$

**and** *PID-ph*:  $\bigwedge\ cid.\ PID \in \in paperIDs\ s\ cid \implies phase\ s\ cid < disPH$

**using** *reachNT-reach* **unfolding**  $\Delta 1$ -def *B*-def **by** *auto*

**show** *iaction*  $?\Delta\ s\ vl\ s1\ vl1 \vee$

$((vl = \square \longrightarrow vl1 = \square) \wedge reaction\ ?\Delta\ s\ vl\ s1\ vl1)$  (**is** *?iact*  $\vee$  ( $\neg \wedge ?react$ ))

**proof**–

**have** *?react* **proof**

**fix**  $a :: act$  **and**  $ou :: out$  **and**  $s' :: state$  **and**  $vl'$

**let**  $?trn = Trans\ s\ a\ ou\ s'$  **let**  $?trn1 = Trans\ s1\ a\ ou\ s'$

**assume** *step*: *step*  $s\ a = (ou, s')$  **and**  $T$ :  $\neg T\ ?trn$  **and**  $c$ : *consume*  $?trn\ vl\ vl'$

```

have  $\varphi: \neg \varphi$  ?trn
proof (cases a)
  case (UUact x3)
  then show ?thesis
    using step PID-ph
    by (cases x3; fastforce simp: uu-defs)
qed auto
hence  $vl': vl' = vl$  using c unfolding consume-def by auto
show match ? $\Delta$  s s1 vl1 a ou s' vl'  $\vee$  ignore ? $\Delta$  s s1 vl1 a ou s' vl' (is ?match
 $\vee$  ?ignore)
proof-
  have ?match proof
    show validTrans ?trn1 unfolding ss1 using step by simp
  next
    show consume ?trn1 vl1 vl1 unfolding consume-def ss1 using  $\varphi$  by auto
  next
    show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
  next
    assume  $\gamma$  ?trn thus  $g$  ?trn =  $g$  ?trn1 unfolding ss1 by simp
  next
    show ? $\Delta$  s' vl' s' vl1
  proof(cases  $\exists$  cid. PID  $\in \in$  paperIDs s cid)
    case False note PID = False
    have ph-PID':  $\bigwedge$  cid. PID  $\in \in$  paperIDs s' cid  $\implies$  phase s' cid < disPH
using PID step rs
  subgoal apply(cases a)
    subgoal for x1 apply(cases x1) apply(fastforce simp: c-defs)+ .
    subgoal for x2 apply(cases x2) apply(fastforce simp: u-defs)+ .
    subgoal for x3 apply(cases x3) apply(fastforce simp: uu-defs)+ .
  by auto
done
hence  $\Delta 1$  s' vl' s' vl1 unfolding  $\Delta 1$ -def B-def vl' using ph-PID' vl by
auto
  thus ?thesis by auto
next
  case True then obtain CID where PID: PID  $\in \in$  paperIDs s CID by
auto
  hence ph: phase s CID < disPH (is ?ph < -) using PID-ph by auto
  show ?thesis
  proof(cases phase s' CID < disPH)
    case True note ph' = True
    show ?thesis proof(cases PID  $\in \in$  paperIDs s' CID)
      case False
      hence  $\Delta 1$  s' vl' s' vl1 unfolding  $\Delta 1$ -def B-def vl' using vl ph' apply
auto
        by (metis PID paperIDs-mono step)
        thus ?thesis by auto
    next
      case True

```

```

      hence  $\Delta 1$   $s'$   $vl'$   $s'$   $vl1$  unfolding  $\Delta 1$ -def B-def  $vl'$  using  $vl$   $ph'$  apply
auto
      by (metis reach-PairI paperIDs-equals rs step)
      thus ?thesis by auto
    qed
  next
    case False
    hence  $ph'$ : phase  $s'$  CID = disPH  $\wedge$  PID  $\in\in$  paperIDs  $s'$  CID
    using PID  $ph$  step rs
    subgoal apply(cases a)
      subgoal for  $x1$  apply(cases  $x1$ ) apply(fastforce simp: c-defs)+ .
      subgoal for  $x2$  apply(cases  $x2$ ) apply(fastforce simp: u-defs)+ .
      subgoal for  $x3$  apply(cases  $x3$ ) apply(fastforce simp: uu-defs)+ .
    by auto
  done
  show ?thesis
  proof(cases  $\exists uid. isPC s' CID uid \wedge pref s' uid PID \neq Conflict$ )
    case True
    hence  $\Delta 2$   $s'$   $vl'$   $s'$   $vl1$  unfolding  $\Delta 2$ -def  $vl'$  using  $vl$   $ph'$  by auto
    thus ?thesis by auto
  next
    case False
    hence  $\Delta e$   $s'$   $vl'$   $s'$   $vl1$  unfolding  $\Delta e$ -def  $vl'$  using  $vl$   $ph'$  by auto
    thus ?thesis by auto
  qed
qed
qed
qed
qed
thus ?thesis by simp
qed
qed
thus ?thesis using  $vl$  by auto
qed
qed

```

**lemma** unwind-cont- $\Delta 2$ : unwind-cont  $\Delta 2$   $\{\Delta 2, \Delta 3, \Delta e\}$

**proof**(rule, simp)

let  $?\Delta = \lambda s vl s1 vl1. \Delta 2 s vl s1 vl1 \vee \Delta 3 s vl s1 vl1 \vee \Delta e s vl s1 vl1$

fix  $s s1 :: state$  and  $vl vl1 :: value list$

assume  $rsT$ : reachNT  $s$  and  $rs1$ : reach  $s1$  and  $\Delta 2 s vl s1 vl1$

then obtain CID uid where uid: isPC  $s$  CID uid pref  $s$  uid PID  $\neq Conflict$

and  $rs$ : reach  $s$  and  $ph$ : phase  $s$  CID = disPH (is  $ph = -$ )

and PID: PID  $\in\in$  paperIDs  $s$  CID and  $ss1$ : eqExcPID  $s s1$

using reachNT-reach unfolding  $\Delta 2$ -def by auto

hence uid-notin: uid  $\notin$  UIDs using  $ph$  reachNT-non-isPC-isChair[OF  $rsT$ ] by force

show iaction  $?\Delta s vl s1 vl1 \vee$

(( $vl = [] \longrightarrow vl1 = []$ )  $\wedge$  reaction  $?\Delta s vl s1 vl1$ ) (is  $?iact \vee (- \wedge ?react)$ )

**proof**(cases  $vl1$ )

```

case (Cons v1 vl1) note vl1 = Cons
have uid1: isPC s1 CID uid pref s1 uid PID ≠ Conflict
using ss1 uid unfolding eqExcPID-def by auto
define a1 where a1 ≡ UUact (uuDis CID uid (pass s uid) PID v1)
obtain s1' ou1 where step1: step s1 a1 = (ou1, s1') by (metis prod.exhaust)
let ?trn1 = Trans s1 a1 ou1 s1'
have s1s1': eqExcPID s1 s1' using a1-def step1 UUact-uuDis-step-eqExcPID
by auto
have ss1': eqExcPID s s1' using eqExcPID-trans[OF ss1 s1s1'] .
hence many-s1': PID ∈∈ paperIDs s1' CID isPC s1' CID uid
pref s1' uid PID ≠ Conflict phase s1' CID = disPH
pass s1' uid = pass s uid
using uid PID ph unfolding eqExcPID-def by auto
hence more-s1': uid ∈∈ userIDs s1' CID ∈∈ confIDs s1'
by (metis paperIDs-confIDs reach-PairI roles-userIDs rs1 step1 many-s1'(1))+
have f: f ?trn1 = v1 unfolding a1-def by simp
have rs1': reach s1' using rs1 step1 by (auto intro: reach-PairI)
have ou1: ou1 = outOK
using step1 uid1 ph unfolding a1-def by (auto simp add: uu-defs many-s1'
more-s1')
have ?iact proof
  show step s1 a1 = (ou1, s1') by fact
next
  show φ: φ ?trn1 using ou1 unfolding a1-def by simp
  thus consume ?trn1 vl1 vl1' using f unfolding consume-def vl1 by simp
next
  show ¬ γ ?trn1 by (simp add: a1-def uid-notin)
next
  have Δ2 s vl s1' vl1' unfolding Δ2-def using ph PID ss1' uid by auto
  thus ?Δ s vl s1' vl1' by simp
qed
thus ?thesis by auto
next
case Nil note vl1 = Nil
have ?react proof
  fix a :: act and ou :: out and s' :: state and vl'
  let ?trn = Trans s a ou s'
  let ?ph' = phase s' CID
  assume step: step s a = (ou, s') and T: ¬ T ?trn and c: consume ?trn vl vl'
  have PID': PID ∈∈ paperIDs s' CID using PID rs by (metis paperIDs-mono
step)
  have uid': isPC s' CID uid ∧ pref s' uid PID ≠ Conflict
  using uid step rs ph PID pref-Conflict-disPH isPC-persistent by auto
  show match ?Δ s s1 vl1 a ou s' vl' ∨ ignore ?Δ s s1 vl1 a ou s' vl' (is ?match
∨ ?ignore)
  proof(cases φ ?trn)
    case False note φ = False
    have vl: vl' = vl using c φ unfolding consume-def by (cases vl) auto
    obtain ou1 and s1' where step1: step s1 a = (ou1, s1') by (metis

```

```

prod.exhaust)
  let ?trn1 = Trans s1 a ou1 s1'
  have s's1': eqExcPID s' s1' using eqExcPID-step[OF ss1 step step1] .
  have  $\varphi 1$ :  $\neg \varphi$  ?trn1 using  $\varphi$  unfolding eqExcPID-step- $\varphi$ [OF ss1 step step1]
  .
  have ?match proof
    show validTrans ?trn1 using step1 by simp
  next
    show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi 1$  by auto
  next
    show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
  next
    assume  $\gamma$  ?trn thus  $g$  ?trn =  $g$  ?trn1
    using eqExcPID-step-out[OF ss1 step step1 rsT rs1 PID] by simp
  next
    show ? $\Delta$  s' vl' s1' vl1
    proof(cases ?ph' = disPH)
      case True
        hence  $\Delta 2$  s' vl' s1' vl1 using PID' s's1' uid' unfolding  $\Delta 2$ -def by
auto
      thus ?thesis by auto
    next
      case False hence ?ph' > disPH
        using ph rs step by (metis le-less phase-increases)
        hence  $\Delta 3$  s' vl' s1' vl1 using s's1' PID' unfolding  $\Delta 3$ -def vl1 by auto
        thus ?thesis by auto
    qed
  qed
  thus ?thesis by simp
next
case True note  $\varphi = \text{True}$ 
  have s's: eqExcPID s' s using eqExcPID-sym[OF  $\varphi$ -step-eqExcPID[OF  $\varphi$ 
step]] .
  have s's1: eqExcPID s' s1 using eqExcPID-trans[OF s's ss1] .
  have ?ignore proof
    show  $\neg \gamma$  ?trn using T- $\varphi$ - $\gamma$   $\varphi$  rsT step by auto
  next
    show ? $\Delta$  s' vl' s1 vl1
    proof(cases ?ph' = disPH)
      case True
        hence  $\Delta 2$  s' vl' s1 vl1 using s's1 PID' uid' unfolding  $\Delta 2$ -def by auto
        thus ?thesis by auto
    next
      case False hence ?ph' > disPH
        using ph rs step by (metis le-less phase-increases)
        hence  $\Delta 3$  s' vl' s1 vl1 using s's1 PID' unfolding  $\Delta 3$ -def vl1 by auto
        thus ?thesis by auto
    qed
  qed

```

```

    thus ?thesis by auto
  qed
  qed
  thus ?thesis using vl1 by auto
  qed
  qed

lemma unwind-cont- $\Delta\mathcal{B}$ : unwind-cont  $\Delta\mathcal{B}$   $\{\Delta\mathcal{B},\Delta e\}$ 
proof(rule, simp)
  let  $\Delta = \lambda s vl s1 vl1. \Delta\mathcal{B} s vl s1 vl1 \vee \Delta e s vl s1 vl1$ 
  fix  $s s1 :: state$  and  $vl vl1 :: value list$ 
  assume  $rsT: reachNT s$  and  $rs1: reach s1$  and  $\Delta\mathcal{B} s vl s1 vl1$ 
  then obtain  $CID$  where  $rs: reach s$  and  $ph: phase s CID > disPH$  (is ?ph <
-)
  and  $PID: PID \in\in paperIDs s CID$ 
  and  $ss1: eqExcPID s s1$  and  $vl1: vl1 = []$ 
  using reachNT-reach unfolding  $\Delta\mathcal{B}$ -def by auto
  show iaction  $\Delta s vl s1 vl1 \vee$ 
    ( $(vl = [] \longrightarrow vl1 = []) \wedge reaction \Delta s vl s1 vl1$ ) (is ?iact  $\vee$  ( $- \wedge ?react$ ))
  proof-
    have ?react
    proof
      fix  $a :: act$  and  $ou :: out$  and  $s' :: state$  and  $vl'$ 
      let ?trn = Trans  $s a ou s'$ 
      let ?ph' = phase  $s' CID$ 
      assume step: step  $s a = (ou, s')$  and  $T: \neg T ?trn$  and  $c: consume ?trn vl vl'$ 
      have  $PID': PID \in\in paperIDs s' CID$  using  $PID rs$  by (metis paperIDs-mono
step)
      have  $ph': phase s' CID > disPH$  using  $ph rs$  by (meson less-le-trans local.step
phase-increases)
      show match  $\Delta s s1 vl1 a ou s' vl' \vee ignore \Delta s s1 vl1 a ou s' vl'$  (is ?match
 $\vee ?ignore$ )
      proof-
        have  $\varphi: \neg \varphi ?trn$  using  $ph step$  unfolding  $\varphi$ -def2 apply (auto simp:
uu-defs)
        using  $PID less-not-refl2 paperIDs-equals rs$  by blast
        have  $vl: vl' = vl$  using  $c \varphi$  unfolding consume-def by (cases vl) auto
        obtain  $ou1$  and  $s1'$  where step1: step  $s1 a = (ou1, s1')$  by (metis
prod.exhaust)
        let ?trn1 = Trans  $s1 a ou1 s1'$ 
        have  $s's1': eqExcPID s' s1'$  using eqExcPID-step[OF ss1 step step1] .
        have  $\varphi1: \neg \varphi ?trn1$  using  $\varphi$  unfolding eqExcPID-step- $\varphi$ [OF ss1 step step1]
.

      have ?match proof
        show validTrans ?trn1 using step1 by simp
      next
      show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi1$  by auto
      next
      show  $\gamma ?trn = \gamma ?trn1$  unfolding ss1 by simp
    end
  end

```

```

next
  assume  $\gamma$  ?trn thus  $g$  ?trn =  $g$  ?trn1
  using eqExcPID-step-out[OF ss1 step step1 rsT rs1 PID] by simp
next
  have  $\Delta \exists s' vl' s1' vl1$  using  $ph'$  PID'  $s's1'$  unfolding  $\Delta \exists$ -def vl1 by
auto
  thus ? $\Delta$   $s' vl' s1' vl1$  by simp
qed
thus ?thesis by simp
qed
qed
thus ?thesis using vl1 by simp
qed
qed

```

**definition** *K1exit* where

```

K1exit cid  $s \equiv$ 
( $PID \in \in$  paperIDs  $s$  cid  $\wedge$  phase  $s$  cid  $\geq$  disPH  $\wedge$ 
 $\neg (\exists uid. isPC s cid uid \wedge pref s uid PID \neq Conflict)$ )

```

**lemma** *invarNT-K1exit*: invarNT (*K1exit* cid)

**unfolding** *invarNT-def* **apply** (safe dest!: reachNT-reach)

```

  subgoal for - a apply(cases a)
    subgoal for x1 apply(cases x1) apply (fastforce simp add: c-defs K1exit-def
geq-noPH-confIDs)+ .
    subgoal for x2
      apply(cases x2)
        apply (fastforce simp add: u-defs K1exit-def paperIDs-equals)
        apply (fastforce simp add: u-defs K1exit-def paperIDs-equals)
        apply (fastforce simp add: u-defs K1exit-def paperIDs-equals)
        apply (fastforce simp add: u-defs K1exit-def paperIDs-equals)
        apply (fastforce simp add: u-defs K1exit-def paperIDs-equals)
        subgoal for x61 apply(cases x61 = cid)
          apply (fastforce simp add: u-defs K1exit-def paperIDs-equals)+ .
          apply (fastforce simp add: u-defs K1exit-def paperIDs-equals) .
        subgoal for x3 apply(cases x3) apply (fastforce simp add: uu-defs K1exit-def)+
.
    by auto
done

```

**lemma** *noVal-K1exit*: noVal (*K1exit* cid)  $v$

**apply**(rule no $\varphi$ -noVal)

**unfolding** no $\varphi$ -def **apply** safe

```

  subgoal for - a apply(cases a)
    apply (fastforce simp add: c-defs K1exit-def)
    apply (fastforce simp add: c-defs K1exit-def)
  subgoal for x3
    apply(cases x3) apply (auto simp add: uu-defs K1exit-def)

```

```

    apply (metis paperIDs-equals reachNT-reach) .
  by auto
done

lemma unwind-exit- $\Delta e$ : unwind-exit  $\Delta e$ 
proof
  fix s s1 :: state and vl vl1 :: value list
  assume rsT: reachNT s and rs1: reach s1 and  $\Delta e$ :  $\Delta e$  s vl s1 vl1
  hence vl: vl  $\neq$  [] using reachNT-reach unfolding  $\Delta e$ -def by auto
  then obtain CID where K1exit CID s using  $\Delta e$  unfolding K1exit-def  $\Delta e$ -def
  by auto
  thus vl  $\neq$  []  $\wedge$  exit s (hd vl) apply(simp add: vl)
  by (metis rsT exitI2 invarNT-K1exit noVal-K1exit)
qed

theorem secure: secure
apply(rule unwind-decomp3-secure[of  $\Delta 1$   $\Delta 2$   $\Delta e$   $\Delta 3$ ])
using
init- $\Delta 1$ 
unwind-cont- $\Delta 1$  unwind-cont- $\Delta 2$  unwind-cont- $\Delta 3$ 
unwind-exit- $\Delta e$ 
by auto

end
theory Discussion-All
imports
Discussion-NCPC
begin

end
theory Decision-Intro
imports ../Safety-Properties
begin

```

## 8 Decision Confidentiality

In this section, we prove confidentiality properties for the accept-reject decision of papers submitted to a conference. The secrets (values) of interest are therefore the different updates of the decision of a given paper with id PID.

Here, we have two points of compromise between the bound and the trigger (which yield two properties). Let

- T1 denote “PC membership having no conflict with that paper and the conference having moved to the discussion stage”
- T2 denote “PC membership or authorship, and the conference having

moved to the notification phase”

The two trigger-bound combinations are:

- weak trigger (T1 or T2) paired with strong bound (allowing to learn almost nothing)
- strong trigger (T1) paired with weak bound (allowing to learn the last updated version of the decision)

```
end  
theory Decision-Value-Setup  
imports Decision-Intro  
begin
```

The ID of the paper under scrutiny:

```
consts PID :: paperID
```

## 8.1 Preliminaries

```
declare updates-commute-paper[simp]
```

```
fun eqExcD :: paper  $\Rightarrow$  paper  $\Rightarrow$  bool where  
eqExcD (Paper title abstract ct reviews dis decs)  
  (Paper title1 abstract1 ct1 reviews1 dis1 decs1) =  
  (title = title1  $\wedge$  abstract = abstract1  $\wedge$  ct = ct1  $\wedge$  reviews = reviews1  $\wedge$  dis =  
  dis1)
```

```
lemma eqExcD:  
eqExcD pap pap1 =  
  (titlePaper pap = titlePaper pap1  $\wedge$  abstractPaper pap = abstractPaper pap1  $\wedge$   
  contentPaper pap = contentPaper pap1  $\wedge$   
  reviewsPaper pap = reviewsPaper pap1  $\wedge$  disPaper pap = disPaper pap1)  
by(cases pap, cases pap1, auto)
```

```
lemma eqExcD-eq[simp,intro!]: eqExcD pap pap  
by(cases pap) auto
```

```
lemma eqExcD-sym:  
assumes eqExcD pap pap1  
shows eqExcD pap1 pap  
apply(cases pap, cases pap1)  
using assms by auto
```

```
lemma eqExcD-trans:  
assumes eqExcD pap pap1 and eqExcD pap1 pap2  
shows eqExcD pap pap2  
apply(cases pap, cases pap1, cases pap2)
```

**using** *assms* **by** *auto*

**definition** *eeqExcPID* **where**

*eeqExcPID* *paps paps1*  $\equiv$

$\forall$  *pid*. if *pid* = *PID* then *eqExcD* (*paps pid*) (*paps1 pid*) else *paps pid* = *paps1 pid*

**lemma** *eeqExcPID-eeq[simp,intro!]*: *eeqExcPID* *s s*

**unfolding** *eeqExcPID-def* **by** *auto*

**lemma** *eeqExcPID-sym*:

**assumes** *eeqExcPID* *s s1* **shows** *eeqExcPID* *s1 s*

**using** *assms* *eqExcD-sym* **unfolding** *eeqExcPID-def* **by** *auto*

**lemma** *eeqExcPID-trans*:

**assumes** *eeqExcPID* *s s1* **and** *eeqExcPID* *s1 s2* **shows** *eeqExcPID* *s s2*

**using** *assms* *eqExcD-trans* **unfolding** *eeqExcPID-def* **by** *simp blast*

**lemma** *eeqExcPID-imp*:

*eeqExcPID* *paps paps1*  $\implies$  *eqExcD* (*paps PID*) (*paps1 PID*)

$\llbracket$ *eeqExcPID* *paps paps1*; *pid*  $\neq$  *PID* $\rrbracket \implies$  *paps pid* = *paps1 pid*

**unfolding** *eeqExcPID-def* **by** *auto*

**lemma** *eeqExcPID-cong*:

**assumes** *eeqExcPID* *paps paps1*

**and** *pid* = *PID*  $\implies$  *eqExcD* *uu uu1*

**and** *pid*  $\neq$  *PID*  $\implies$  *uu* = *uu1*

**shows** *eeqExcPID* (*paps* (*pid* := *uu*)) (*paps1* (*pid* := *uu1*))

**using** *assms* **unfolding** *eeqExcPID-def* **by** *auto*

**lemma** *eeqExcPID-RDD*:

*eeqExcPID* *paps paps1*  $\implies$

*titlePaper* (*paps PID*) = *titlePaper* (*paps1 PID*)  $\wedge$

*abstractPaper* (*paps PID*) = *abstractPaper* (*paps1 PID*)  $\wedge$

*contentPaper* (*paps PID*) = *contentPaper* (*paps1 PID*)  $\wedge$

*reviewsPaper* (*paps PID*) = *reviewsPaper* (*paps1 PID*)  $\wedge$

*disPaper* (*paps PID*) = *disPaper* (*paps1 PID*)

**using** *eeqExcPID-def* **unfolding** *eqExcD* **by** *auto*

**definition** *eqExcPID* :: *state*  $\Rightarrow$  *state*  $\Rightarrow$  *bool* **where**

*eqExcPID* *s s1*  $\equiv$

*confIDs* *s* = *confIDs* *s1*  $\wedge$  *conf* *s* = *conf* *s1*  $\wedge$

*userIDs* *s* = *userIDs* *s1*  $\wedge$  *pass* *s* = *pass* *s1*  $\wedge$  *user* *s* = *user* *s1*  $\wedge$  *roles* *s* = *roles* *s1*  $\wedge$

*paperIDs* *s* = *paperIDs* *s1*

$\wedge$

*eeqExcPID* (*paper* *s*) (*paper* *s1*)

$\wedge$

$pref\ s = pref\ s1 \wedge$   
 $voronkov\ s = voronkov\ s1 \wedge$   
 $news\ s = news\ s1 \wedge phase\ s = phase\ s1$

**lemma** *eqExcPID-eq[simp,intro!]*: *eqExcPID s s*  
**unfolding** *eqExcPID-def* **by** *auto*

**lemma** *eqExcPID-sym*:  
**assumes** *eqExcPID s s1* **shows** *eqExcPID s1 s*  
**using** *assms eqExcPID-sym* **unfolding** *eqExcPID-def* **by** *auto*

**lemma** *eqExcPID-trans*:  
**assumes** *eqExcPID s s1* **and** *eqExcPID s1 s2* **shows** *eqExcPID s s2*  
**using** *assms eqExcPID-trans* **unfolding** *eqExcPID-def* **by** *auto*

**lemma** *eqExcPID-imp*:  
 $eqExcPID\ s\ s1 \implies$   
 $confIDs\ s = confIDs\ s1 \wedge conf\ s = conf\ s1 \wedge$   
 $userIDs\ s = userIDs\ s1 \wedge pass\ s = pass\ s1 \wedge user\ s = user\ s1 \wedge roles\ s = roles$   
 $s1 \wedge$   
 $paperIDs\ s = paperIDs\ s1$   
 $\wedge$   
 $eeqExcPID\ (paper\ s)\ (paper\ s1)$   
 $\wedge$   
 $pref\ s = pref\ s1 \wedge$   
 $voronkov\ s = voronkov\ s1 \wedge$   
 $news\ s = news\ s1 \wedge phase\ s = phase\ s1 \wedge$

$getAllPaperIDs\ s = getAllPaperIDs\ s1 \wedge$   
 $isRev\ s\ cid\ uid\ pid = isRev\ s1\ cid\ uid\ pid \wedge$   
 $getReviewIndex\ s\ cid\ uid\ pid = getReviewIndex\ s1\ cid\ uid\ pid \wedge$   
 $getRevRole\ s\ cid\ uid\ pid = getRevRole\ s1\ cid\ uid\ pid$   
**unfolding** *eqExcPID-def getAllPaperIDs-def*  
**unfolding** *isRev-def getReviewIndex-def getRevRole-def* **by** *auto*

**lemma** *eqExcPID-imp1*:  
 $eqExcPID\ s\ s1 \implies eqExcD\ (paper\ s\ pid)\ (paper\ s1\ pid)$   
 $eqExcPID\ s\ s1 \implies pid \neq PID \vee PID \neq pid \implies$   
 $paper\ s\ pid = paper\ s1\ pid \wedge$   
 $getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$   
**unfolding** *eqExcPID-def getNthReview-def eeqExcPID-def*  
**apply** *auto*  
**by** (*metis eqExcD-eq*)

**lemma** *eqExcPID-imp2*:  
**assumes** *eqExcPID s s1* **and**  $pid \neq PID \vee PID \neq pid$   
**shows**  $getReviewersReviews\ s\ cid\ pid = getReviewersReviews\ s1\ cid\ pid$   
**proof** –

**have**  
 $(\lambda uID. \text{if isRev } s \text{ cid } uID \text{ pid then } [(uID, \text{getNthReview } s \text{ pid } (\text{getReviewIndex } s \text{ cid } uID \text{ pid}))] \text{ else } [])) =$   
 $(\lambda uID. \text{if isRev } s1 \text{ cid } uID \text{ pid then } [(uID, \text{getNthReview } s1 \text{ pid } (\text{getReviewIndex } s1 \text{ cid } uID \text{ pid}))] \text{ else } [])$   
**apply**(rule ext)  
**using** *assms* **by** (auto simp: eqExcPID-imp eqExcPID-imp1)  
**thus** ?thesis **unfolding** getReviewersReviews-def **using** *assms* **by** (simp add: eqExcPID-imp)  
**qed**

**lemma** eqExcPID-RDD:

$eqExcPID \ s \ s1 \implies$   
 $titlePaper \ (paper \ s \ PID) = titlePaper \ (paper \ s1 \ PID) \wedge$   
 $abstractPaper \ (paper \ s \ PID) = abstractPaper \ (paper \ s1 \ PID) \wedge$   
 $contentPaper \ (paper \ s \ PID) = contentPaper \ (paper \ s1 \ PID) \wedge$   
 $reviewsPaper \ (paper \ s \ PID) = reviewsPaper \ (paper \ s1 \ PID) \wedge$   
 $disPaper \ (paper \ s \ PID) = disPaper \ (paper \ s1 \ PID)$   
**using** eqExcPID-imp eqExcPID-RDD **by** auto

**lemma** eqExcPID-cong[simp, intro]:

$\bigwedge uu1 \ uu2. eqExcPID \ s \ s1 \implies uu1 = uu2 \implies eqExcPID \ (s \ (\!|confIDs := uu1|)) \ (s1 \ (\!|confIDs := uu2|))$   
 $\bigwedge uu1 \ uu2. eqExcPID \ s \ s1 \implies uu1 = uu2 \implies eqExcPID \ (s \ (\!|conf := uu1|)) \ (s1 \ (\!|conf := uu2|))$

$\bigwedge uu1 \ uu2. eqExcPID \ s \ s1 \implies uu1 = uu2 \implies eqExcPID \ (s \ (\!|userIDs := uu1|)) \ (s1 \ (\!|userIDs := uu2|))$   
 $\bigwedge uu1 \ uu2. eqExcPID \ s \ s1 \implies uu1 = uu2 \implies eqExcPID \ (s \ (\!|pass := uu1|)) \ (s1 \ (\!|pass := uu2|))$   
 $\bigwedge uu1 \ uu2. eqExcPID \ s \ s1 \implies uu1 = uu2 \implies eqExcPID \ (s \ (\!|user := uu1|)) \ (s1 \ (\!|user := uu2|))$   
 $\bigwedge uu1 \ uu2. eqExcPID \ s \ s1 \implies uu1 = uu2 \implies eqExcPID \ (s \ (\!|roles := uu1|)) \ (s1 \ (\!|roles := uu2|))$

$\bigwedge uu1 \ uu2. eqExcPID \ s \ s1 \implies uu1 = uu2 \implies eqExcPID \ (s \ (\!|paperIDs := uu1|)) \ (s1 \ (\!|paperIDs := uu2|))$   
 $\bigwedge uu1 \ uu2. eqExcPID \ s \ s1 \implies eqExcPID \ uu1 \ uu2 \implies eqExcPID \ (s \ (\!|paper := uu1|)) \ (s1 \ (\!|paper := uu2|))$

$\bigwedge uu1 \ uu2. eqExcPID \ s \ s1 \implies uu1 = uu2 \implies eqExcPID \ (s \ (\!|pref := uu1|)) \ (s1 \ (\!|pref := uu2|))$   
 $\bigwedge uu1 \ uu2. eqExcPID \ s \ s1 \implies uu1 = uu2 \implies eqExcPID \ (s \ (\!|voronkov := uu1|)) \ (s1 \ (\!|voronkov := uu2|))$   
 $\bigwedge uu1 \ uu2. eqExcPID \ s \ s1 \implies uu1 = uu2 \implies eqExcPID \ (s \ (\!|news := uu1|)) \ (s1 \ (\!|news := uu2|))$   
 $\bigwedge uu1 \ uu2. eqExcPID \ s \ s1 \implies uu1 = uu2 \implies eqExcPID \ (s \ (\!|phase := uu1|)) \ (s1 \ (\!|phase := uu2|))$

**unfolding** eqExcPID-def **by** auto

```

lemma eqExcPID-Paper:
assumes s's1': eqExcPID s s1
and paper s pid = Paper title abstract content reviews dis decs
and paper s1 pid = Paper title1 abstract1 content1 reviews1 dis1 decs1
shows title = title1 ∧ abstract = abstract1 ∧ content = content1 ∧ reviews =
reviews1 ∧ dis = dis1
using assms unfolding eqExcPID-def apply (auto simp: eqExcD eeqExcPID-def)
by (metis titlePaper.simps abstractPaper.simps contentPaper.simps reviewsPaper.simps
disPaper.simps
    )+

```

Weaker equivalence relations that allow differences in the final decision. This is used for verifying the confidentiality property that only protects earlier updates to the decision.

```

fun eqExcD2 :: paper ⇒ paper ⇒ bool where
eqExcD2 (Paper title abstract ct reviews dis decs )
  (Paper title1 abstract1 ct1 reviews1 dis1 decs1) =
  (title = title1 ∧ abstract = abstract1 ∧ ct = ct1 ∧ reviews = reviews1 ∧ dis =
dis1 ∧
   hd decs = hd decs1)

```

```

lemma eqExcD2:
eqExcD2 pap pap1 =
  (titlePaper pap = titlePaper pap1 ∧ abstractPaper pap = abstractPaper pap1 ∧
   contentPaper pap = contentPaper pap1 ∧
   reviewsPaper pap = reviewsPaper pap1 ∧ disPaper pap = disPaper pap1 ∧
   hd (decsPaper pap) = hd (decsPaper pap1)
  )
by(cases pap, cases pap1, auto)

```

```

lemma eqExcD2-eq[simp,intro!]: eqExcD2 pap pap
by(cases pap) auto

```

```

lemma eqExcD2-sym:
assumes eqExcD2 pap pap1
shows eqExcD2 pap1 pap
apply(cases pap, cases pap1)
using assms by auto

```

```

lemma eqExcD2-trans:
assumes eqExcD2 pap pap1 and eqExcD2 pap1 pap2
shows eqExcD2 pap pap2
apply(cases pap, cases pap1, cases pap2)
using assms by auto

```

```

definition eeqExcPID2 where
eeqExcPID2 paps paps1 ≡

```

$\forall pid.$  if  $pid = PID$  then  $eqExcD2 (paps pid) (paps1 pid)$  else  $paps pid = paps1 pid$

**lemma**  $eeqExcPID2$ - $eeq[simp,intro!]$ :  $eeqExcPID2 s s$   
**unfolding**  $eeqExcPID2$ - $def$  **by** *auto*

**lemma**  $eeqExcPID2$ - $sym$ :  
**assumes**  $eeqExcPID2 s s1$  **shows**  $eeqExcPID2 s1 s$   
**using**  $assms$   $eqExcD2$ - $sym$  **unfolding**  $eeqExcPID2$ - $def$  **by** *auto*

**lemma**  $eeqExcPID2$ - $trans$ :  
**assumes**  $eeqExcPID2 s s1$  **and**  $eeqExcPID2 s1 s2$  **shows**  $eeqExcPID2 s s2$   
**using**  $assms$   $eqExcD2$ - $trans$  **unfolding**  $eeqExcPID2$ - $def$  **by** *simp blast*

**lemma**  $eeqExcPID2$ - $imp$ :  
 $eeqExcPID2 paps paps1 \implies eqExcD2 (paps PID) (paps1 PID)$   
 $\llbracket eeqExcPID2 paps paps1; pid \neq PID \rrbracket \implies paps pid = paps1 pid$   
**unfolding**  $eeqExcPID2$ - $def$  **by** *auto*

**lemma**  $eeqExcPID2$ - $cong$ :  
**assumes**  $eeqExcPID2 paps paps1$   
**and**  $pid = PID \implies eqExcD2 uu uu1$   
**and**  $pid \neq PID \implies uu = uu1$   
**shows**  $eeqExcPID2 (paps (pid := uu)) (paps1 (pid := uu1))$   
**using**  $assms$  **unfolding**  $eeqExcPID2$ - $def$  **by** *auto*

**lemma**  $eeqExcPID2$ - $RDD$ :  
 $eeqExcPID2 paps paps1 \implies$   
 $titlePaper (paps PID) = titlePaper (paps1 PID) \wedge$   
 $abstractPaper (paps PID) = abstractPaper (paps1 PID) \wedge$   
 $contentPaper (paps PID) = contentPaper (paps1 PID) \wedge$   
 $reviewsPaper (paps PID) = reviewsPaper (paps1 PID) \wedge$   
 $disPaper (paps PID) = disPaper (paps1 PID) \wedge$   
 $hd (decsPaper (paps PID)) = hd (decsPaper (paps1 PID))$   
**using**  $eeqExcPID2$ - $def$  **unfolding**  $eqExcD2$  **by** *auto*

**definition**  $eqExcPID2 :: state \Rightarrow state \Rightarrow bool$  **where**  
 $eqExcPID2 s s1 \equiv$   
 $confIDs s = confIDs s1 \wedge conf s = conf s1 \wedge$   
 $userIDs s = userIDs s1 \wedge pass s = pass s1 \wedge user s = user s1 \wedge roles s = roles$   
 $s1 \wedge$   
 $paperIDs s = paperIDs s1$   
 $\wedge$   
 $eeqExcPID2 (paper s) (paper s1)$   
 $\wedge$   
 $pref s = pref s1 \wedge$   
 $voronkov s = voronkov s1 \wedge$   
 $news s = news s1 \wedge phase s = phase s1$

**lemma** *eqExcPID2-eq[simp,intro!]*: *eqExcPID2 s s*  
**unfolding** *eqExcPID2-def* **by** *auto*

**lemma** *eqExcPID2-sym*:  
**assumes** *eqExcPID2 s s1* **shows** *eqExcPID2 s1 s*  
**using** *assms eqExcPID2-sym* **unfolding** *eqExcPID2-def* **by** *auto*

**lemma** *eqExcPID2-trans*:  
**assumes** *eqExcPID2 s s1* **and** *eqExcPID2 s1 s2* **shows** *eqExcPID2 s s2*  
**using** *assms eqExcPID2-trans* **unfolding** *eqExcPID2-def* **by** *auto*

**lemma** *eqExcPID2-imp*:  
*eqExcPID2 s s1*  $\implies$   
*confIDs s = confIDs s1*  $\wedge$  *conf s = conf s1*  $\wedge$   
*userIDs s = userIDs s1*  $\wedge$  *pass s = pass s1*  $\wedge$  *user s = user s1*  $\wedge$  *roles s = roles*  
*s1*  $\wedge$   
*paperIDs s = paperIDs s1*  
 $\wedge$   
*eeqExcPID2 (paper s) (paper s1)*  
 $\wedge$   
*pref s = pref s1*  $\wedge$   
*voronkov s = voronkov s1*  $\wedge$   
*news s = news s1*  $\wedge$  *phase s = phase s1*  $\wedge$

*getAllPaperIDs s = getAllPaperIDs s1*  $\wedge$   
*isRev s cid uid pid = isRev s1 cid uid pid*  $\wedge$   
*getReviewIndex s cid uid pid = getReviewIndex s1 cid uid pid*  $\wedge$   
*getRevRole s cid uid pid = getRevRole s1 cid uid pid*  
**unfolding** *eqExcPID2-def* *getAllPaperIDs-def*  
**unfolding** *isRev-def* *getReviewIndex-def* *getRevRole-def* **by** *auto*

**lemma** *eqExcPID2-imp1*:  
*eqExcPID2 s s1*  $\implies$  *eqExcD2 (paper s pid) (paper s1 pid)*  
*eqExcPID2 s s1*  $\implies$  *pid  $\neq$  PID  $\vee$  PID  $\neq$  pid*  $\implies$   
*paper s pid = paper s1 pid*  $\wedge$   
*getNthReview s pid n = getNthReview s1 pid n*  
**unfolding** *eqExcPID2-def* *getNthReview-def* *eeqExcPID2-def*  
**apply** *auto*  
**by** (*metis eqExcD2-eq*)

**lemma** *eqExcPID2-imp2*:  
**assumes** *eqExcPID2 s s1* **and** *pid  $\neq$  PID  $\vee$  PID  $\neq$  pid*  
**shows** *getReviewersReviews s cid pid = getReviewersReviews s1 cid pid*  
**proof** –  
**have**  
 $(\lambda uID. \text{if } \text{isRev } s \text{ cid } uID \text{ pid then } [(uID, \text{getNthReview } s \text{ pid } (\text{getReviewIndex } s \text{ cid } uID \text{ pid}))] \text{ else } []) =$

```

  (λuID. if isRev s1 cid uID pid then [(uID, getNthReview s1 pid (getReviewIndex
s1 cid uID pid))] else [])
  apply(rule ext)
  using assms by (auto simp: eqExcPID2-imp eqExcPID2-imp1)
  thus ?thesis unfolding getReviewersReviews-def using assms by (simp add:
eqExcPID2-imp)
qed

```

**lemma** *eqExcPID2-RDD*:

```

eqExcPID2 s s1 ⇒
  titlePaper (paper s PID) = titlePaper (paper s1 PID) ∧
  abstractPaper (paper s PID) = abstractPaper (paper s1 PID) ∧
  contentPaper (paper s PID) = contentPaper (paper s1 PID) ∧
  reviewsPaper (paper s PID) = reviewsPaper (paper s1 PID) ∧
  disPaper (paper s PID) = disPaper (paper s1 PID)
using eqExcPID2-imp eqExcPID2-RDD by auto

```

**lemma** *eqExcPID2-cong[simp, intro]*:

```

∧ uu1 uu2. eqExcPID2 s s1 ⇒ uu1 = uu2 ⇒ eqExcPID2 (s (|confIDs := uu1))
(s1 (|confIDs := uu2))
∧ uu1 uu2. eqExcPID2 s s1 ⇒ uu1 = uu2 ⇒ eqExcPID2 (s (|conf := uu1))
(s1 (|conf := uu2))

```

```

∧ uu1 uu2. eqExcPID2 s s1 ⇒ uu1 = uu2 ⇒ eqExcPID2 (s (|userIDs := uu1))
(s1 (|userIDs := uu2))
∧ uu1 uu2. eqExcPID2 s s1 ⇒ uu1 = uu2 ⇒ eqExcPID2 (s (|pass := uu1))
(s1 (|pass := uu2))
∧ uu1 uu2. eqExcPID2 s s1 ⇒ uu1 = uu2 ⇒ eqExcPID2 (s (|user := uu1))
(s1 (|user := uu2))
∧ uu1 uu2. eqExcPID2 s s1 ⇒ uu1 = uu2 ⇒ eqExcPID2 (s (|roles := uu1))
(s1 (|roles := uu2))

```

```

∧ uu1 uu2. eqExcPID2 s s1 ⇒ uu1 = uu2 ⇒ eqExcPID2 (s (|paperIDs :=
uu1)) (s1 (|paperIDs := uu2))
∧ uu1 uu2. eqExcPID2 s s1 ⇒ eqExcPID2 uu1 uu2 ⇒ eqExcPID2 (s (|paper
:= uu1)) (s1 (|paper := uu2))

```

```

∧ uu1 uu2. eqExcPID2 s s1 ⇒ uu1 = uu2 ⇒ eqExcPID2 (s (|pref := uu1))
(s1 (|pref := uu2))
∧ uu1 uu2. eqExcPID2 s s1 ⇒ uu1 = uu2 ⇒ eqExcPID2 (s (|voronkov :=
uu1)) (s1 (|voronkov := uu2))
∧ uu1 uu2. eqExcPID2 s s1 ⇒ uu1 = uu2 ⇒ eqExcPID2 (s (|news := uu1))
(s1 (|news := uu2))
∧ uu1 uu2. eqExcPID2 s s1 ⇒ uu1 = uu2 ⇒ eqExcPID2 (s (|phase := uu1))
(s1 (|phase := uu2))
unfolding eqExcPID2-def by auto

```

**lemma** *eqExcPID2-Paper*:

```

assumes s's1': eqExcPID2 s s1

```

```

and paper s pid = Paper title abstract content reviews dis decs
and paper s1 pid = Paper title1 abstract1 content1 reviews1 dis1 decs1
shows title = title1  $\wedge$  abstract = abstract1  $\wedge$  content = content1  $\wedge$  reviews =
reviews1  $\wedge$ 
  dis = dis1
using assms unfolding eqExcPID2-def apply (auto simp: eqExcD2 eeqExcPID2-def)
  by (metis titlePaper.simps abstractPaper.simps contentPaper.simps reviewsPa-
per.simps
disPaper.simps)+

```

## 8.2 Value Setup

```

type-synonym value = decision

```

```

fun  $\varphi$  :: (state,act,out) trans  $\Rightarrow$  bool where
 $\varphi$  (Trans - (UUact (uuDec cid uid p pid dec)) ou -) = (pid = PID  $\wedge$  ou = outOK)
|
 $\varphi$  - = False

```

**lemma**  $\varphi$ -def2:

```

 $\varphi$  (Trans s a ou s') = ( $\exists$  cid uid p dec. a = UUact (uuDec cid uid p PID dec)  $\wedge$ 
ou = outOK)

```

**proof** (cases a)

**case** (UUact x3)

**then show** ?thesis **by** (cases x3; auto)

**qed** auto

```

fun f :: (state,act,out) trans  $\Rightarrow$  value where
f (Trans - (UUact (uuDec cid uid p pid dec)) - -) = dec

```

**lemma** UUact-uuDec-step-eqExcPID:

**assumes** a: a = UUact (uuDec cid uid p PID dec)

**and** step s a = (ou,s')

**shows** eqExcPID s s'

**using** *assms* **unfolding** eqExcPID-def eeqExcPID-def **by** (auto simp: uu-defs)

**lemma**  $\varphi$ -step-eqExcPID:

**assumes**  $\varphi$ :  $\varphi$  (Trans s a ou s')

**and** s: step s a = (ou,s')

**shows** eqExcPID s s'

**using**  $\varphi$  UUact-uuDec-step-eqExcPID[OF - s] **unfolding**  $\varphi$ -def2 **by** blast

**lemma** eqExcPID-step:

**assumes** s's1': eqExcPID s s1

**and** step: step s a = (ou,s')

**and** step1: step s1 a = (ou1,s1')

**shows** eqExcPID s' s1'

**proof** -

**note** eqs = eqExcPID-imp[OF s's1']

**note**  $eqs' = eqExcPID-imp1[OF\ s's1']$   
**note**  $simps[simp] = c-defs\ u-defs\ uu-defs\ r-defs\ l-defs\ Paper-dest-conv\ eqExcPID-def\ eqExcPID-def\ eqExcD$   
**note**  $* = step\ step1\ eqs\ eqs'$   
**then show** *?thesis*  
**proof** (cases a)  
  **case** (Cact x1)  
    **then show** *?thesis* **using** \* **by** (cases x1; auto)  
  **next**  
    **case** (Uact x2)  
      **then show** *?thesis* **using** \* **by** (cases x2; auto)  
  **next**  
    **case** (UUact x3)  
      **then show** *?thesis* **using** \* **by** (cases x3; auto)  
**qed** auto  
**qed**

**lemma** *eqExcPID-step-φ-imp*:  
**assumes**  $s's1'$ :  $eqExcPID\ s\ s1$   
**and**  $step$ :  $step\ s\ a = (ou, s')$  **and**  $step1$ :  $step\ s1\ a = (ou1, s1')$   
**and**  $\varphi$ :  $\varphi\ (Trans\ s\ a\ ou\ s')$   
**shows**  $\varphi\ (Trans\ s1\ a\ ou1\ s1')$   
**using** *assms* **unfolding**  $\varphi-def2$  **by** (auto *simp* *add*: *uu-defs* *eqExcPID-imp*)

**lemma** *eqExcPID-step-φ*:  
**assumes**  $s's1'$ :  $eqExcPID\ s\ s1$   
**and**  $step$ :  $step\ s\ a = (ou, s')$  **and**  $step1$ :  $step\ s1\ a = (ou1, s1')$   
**shows**  $\varphi\ (Trans\ s\ a\ ou\ s') = \varphi\ (Trans\ s1\ a\ ou1\ s1')$   
**by** (*metis* *eqExcPID-step-φ-imp* *eqExcPID-sym* *assms*)

**lemma** *eqExcPID2-step*:  
**assumes**  $s's1'$ :  $eqExcPID2\ s\ s1$   
**and**  $step$ :  $step\ s\ a = (ou, s')$   
**and**  $step1$ :  $step\ s1\ a = (ou1, s1')$   
**shows**  $eqExcPID2\ s'\ s1'$   
**proof** –  
  **note**  $eqs = eqExcPID2-imp[OF\ s's1']$   
  **note**  $eqs' = eqExcPID2-imp1[OF\ s's1']$

**note**  $simps[simp] = c-defs\ u-defs\ uu-defs\ r-defs\ l-defs\ Paper-dest-conv\ eqExcPID2-def\ eqExcPID2-def\ eqExcD2$   
**note**  $* = s's1'\ step\ step1\ eqs\ eqs'$

**then show** *?thesis*

```

proof (cases a)
  case (Cact x1)
    then show ?thesis using * by (cases x1; auto)
  next
    case (Uact x2)
      then show ?thesis using * by (cases x2; auto)
    next
      case (UUact x3)
        then show ?thesis using * by (cases x3; auto)
    qed auto
qed

```

```

lemma eqExcPID2-step-φ-imp:
assumes s's1': eqExcPID2 s s1
and step: step s a = (ou,s') and step1: step s1 a = (ou1,s1')
and φ: φ (Trans s a ou s')
shows φ (Trans s1 a ou1 s1')
using assms unfolding φ-def2 by (auto simp add: uu-defs eqExcPID2-imp)

```

```

lemma eqExcPID2-step-φ:
assumes s's1': eqExcPID2 s s1
and step: step s a = (ou,s') and step1: step s1 a = (ou1,s1')
shows φ (Trans s a ou s') = φ (Trans s1 a ou1 s1')
by (metis eqExcPID2-step-φ-imp eqExcPID2-sym assms)

```

**end**

**theory** Decision-NCPC

**imports** ../Observation-Setup Decision-Value-Setup Bounded-Deducibility-Security.Compositional-Reasoning  
**begin**

### 8.3 Confidentiality protection from non-PC-members

We verify the following property:

A group of users UIDs learn nothing about the various updates of a paper's decision except for the last edited version unless/until a user in UIDs becomes PC member in the paper's conference having no conflict with that paper and the conference moves to the decision stage.

```

fun T :: (state,act,out) trans ⇒ bool where
  T (Trans - - ou s') =
    (∃ uid ∈ UIDs. ∃ cid.
      PID ∈∈ paperIDs s' cid ∧ isPC s' cid uid ∧ pref s' uid PID ≠ Conflict ∧
      phase s' cid ≥ disPH
    )

```

**declare**  $T.simps$  [ $simp$  del]

**definition**  $B :: value\ list \Rightarrow value\ list \Rightarrow bool$  **where**  
 $B\ vl\ vl1 \equiv vl \neq [] \wedge vl1 \neq [] \wedge last\ vl = last\ vl1$

**interpretation**  $BD\text{-}Security\text{-}IO$  **where**  
 $istate = istate$  **and**  $step = step$  **and**  
 $\varphi = \varphi$  **and**  $f = f$  **and**  $\gamma = \gamma$  **and**  $g = g$  **and**  $T = T$  **and**  $B = B$   
**done**

**lemma**  $reachNT\text{-}non\text{-}isPC\text{-}isChair$ :  
**assumes**  $reachNT\ s$  **and**  $uid \in UIDs$   
**shows**

$(PID \in \in paperIDs\ s\ cid \wedge isPC\ s\ cid\ uid \wedge phase\ s\ cid \geq disPH \longrightarrow pref\ s\ uid$   
 $PID = Conflict) \wedge$   
 $(PID \in \in paperIDs\ s\ cid \wedge isChair\ s\ cid\ uid \wedge phase\ s\ cid \geq disPH \longrightarrow pref\ s$   
 $uid\ PID = Conflict)$   
**using**  $assms$   
**apply**  $induct$   
**apply**  $(auto\ simp: istate\text{-}def) []$   
**apply**  $(intro\ conjI)$   
**subgoal for**  $trn$  **apply**  $(cases\ trn, auto\ simp: T.simps\ reachNT\text{-}reach) []$  .  
**by**  $(metis\ T.elims(3)\ isChair\text{-}isPC\ reachNT\text{-}reach\ reach.Step\ tgtOf\text{-}simps)$

**lemma**  $T\text{-}\varphi\text{-}\gamma$ :  
**assumes**  $1: reachNT\ s$  **and**  $2: step\ s\ a = (ou, s') \varphi (Trans\ s\ a\ ou\ s')$   
**shows**  $\neg \gamma (Trans\ s\ a\ ou\ s')$   
**using**  $reachNT\text{-}non\text{-}isPC\text{-}isChair[OF\ 1]$   $2$  **unfolding**  $T.simps\ \varphi\text{-}def2$   
**by**  $(fastforce\ simp\ add: uu\text{-}defs)$

**lemma**  $eqExcPID2\text{-}eqExcPID$ :  
 $eqExcPID2\ s\ s1 \implies eqExcPID\ s\ s1$   
**unfolding**  $eqExcPID\text{-}def\ eqExcPID2\text{-}def\ eqExcPID\text{-}def\ eqExcPID2\text{-}def\ eqExcD2$   
 $eqExcD$  **by**  $auto$

**lemma**  $eqExcPID\text{-}step\text{-}out$ :  
**assumes**  $s's1'$ :  $eqExcPID\ s\ s1$   
**and**  $step$ :  $step\ s\ a = (ou, s')$  **and**  $step1$ :  $step\ s1\ a = (ou1, s1')$   
**and**  $sT$ :  $reachNT\ s$  **and**  $s1$ :  $reach\ s1$   
**and**  $PID$ :  $PID \in \in paperIDs\ s\ cid$   
**and**  $ph$ :  $phase\ s\ cid = disPH$   
**and**  $UIDs$ :  $userOfA\ a \in UIDs$   
**shows**  $ou = ou1$   
**proof** –  
**note**  $Inv = reachNT\text{-}non\text{-}isPC\text{-}isChair[OF\ sT\ UIDs]$   
**note**  $eqs = eqExcPID\text{-}imp[OF\ s's1']$   
**note**  $eqs' = eqExcPID\text{-}imp1[OF\ s's1']$   
**note**  $s = reachNT\text{-}reach[OF\ sT]$

**note**  $\text{simps}[simp] = c\text{-defs } u\text{-defs } uu\text{-defs } r\text{-defs } l\text{-defs } \text{Paper-dest-conv } eqExc\text{-}cPID\text{-def } eqExcPID\text{-def } eqExcD$

**note**  $*$  =  $\text{step } step1 \text{ eqs } eqs' \text{ s } s1 \text{ PID } \text{UIDs } ph \text{ paperIDs-equals}[OF \text{ s}] \text{ Inv}$

**show**  $?thesis$   
**proof** (cases a)  
  **case** (Cact x1)  
  **then show**  $?thesis$  **using**  $*$  **by** (cases x1) *auto*  
**next**  
  **case** (Uact x2)  
  **then show**  $?thesis$  **using**  $*$  **by** (cases x2) *auto*  
**next**  
  **case** (UUact x3)  
  **then show**  $?thesis$  **using**  $*$  **by** (cases x3) *auto*  
**next**  
  **case** (Ract x4)  
  **show**  $?thesis$   
  **proof** (cases x4)  
  **case** (rMyReview x81 x82 x83 x84)  
  **then show**  $?thesis$  **using**  $*$  **Ract by** (auto simp add: getNthReview-def)  
  **next**  
  **case** (rReviews x91 x92 x93 x94)  
  **then show**  $?thesis$  **using**  $*$  **Ract by** (clarsimp; metis eqExcPID-imp2 s's1')  
  **next**  
  **case** (rDecs x101 x102 x103 x104)  
  **then show**  $?thesis$  **using**  $*$  **Ract by** (clarsimp; metis)  
  **next**  
  **case** (rFinalDec x131 x132 x133 x134)  
  **then show**  $?thesis$  **using**  $*$  **Ract by** (clarsimp; metis Suc-n-not-le-n)  
  **qed** (use  $*$  **Ract in** *auto*)  
**next**  
  **case** (Lact x5)  
  **then show**  $?thesis$  **using**  $*$  **by** (cases x5; *auto*; presburger)  
**qed**  
**qed**

**lemma**  $eqExcPID2\text{-step-out}$ :  
**assumes**  $ss1: eqExcPID2 \text{ s } s1$   
**and**  $step: step \text{ s } a = (ou, s')$  **and**  $step1: step \text{ s1 } a = (ou1, s1')$   
**and**  $sT: reachNT \text{ s } \text{and } s1: reach \text{ s1}$   
**and**  $PID: PID \in \in \text{paperIDs } \text{s } cid$   
**and**  $ph: phase \text{ s } cid \geq disPH$   
**and**  $UIDs: userOfA \text{ a } \in \text{UIDs}$   
**and**  $decs\text{-exit}: decsPaper (\text{paper } \text{s } PID) \neq [] \text{ decsPaper } (\text{paper } \text{s1 } PID) \neq []$   
**shows**  $ou = ou1$   
**proof** –  
  **note**  $Inv = reachNT\text{-non-isPC-isChair}[OF \text{ sT } \text{UIDs}]$   
  **note**  $eqs = eqExcPID2\text{-imp}[OF \text{ ss1}]$   
  **note**  $eqs' = eqExcPID2\text{-imp1}[OF \text{ ss1}]$

```

note  $s = \text{reachNT-reach}[OF\ sT]$ 

have  $PID \in \in \text{paperIDs}\ s1\ cid$  using  $PID\ ss1$  unfolding  $\text{eqExcPID2-def}$  by  $\text{auto}$ 
hence  $\text{decs-exit}' : \text{decsPaper}\ (\text{paper}\ s'\ PID) \neq []\ \text{decsPaper}\ (\text{paper}\ s1'\ PID) \neq []$ 
using  $\text{nonempty-decsPaper-persist}\ s\ s1\ PID\ \text{decs-exit}\ \text{step}\ \text{step1}$  by  $\text{metis+}$ 

note  $\text{simps}[simp] = c\text{-defs}\ u\text{-defs}\ uu\text{-defs}\ r\text{-defs}\ l\text{-defs}\ \text{Paper-dest-conv}\ \text{eqExcPID2-def}\ \text{eeqExcPID2-def}\ \text{eqExcD2}$ 
note  $*$  =  $\text{step}\ \text{step1}\ \text{eqs}\ \text{eqs}'\ s\ s1\ PID\ \text{UIDs}\ \text{ph}\ \text{paperIDs-equals}[OF\ s]\ \text{Inv}$ 

show  $?thesis$ 
proof ( $\text{cases}\ a$ )
  case ( $\text{Cact}\ x1$ )
    then show  $?thesis$  using  $*$  by ( $\text{cases}\ x1$ )  $\text{auto}$ 
  next
    case ( $\text{Uact}\ x2$ )
      then show  $?thesis$  using  $*$  by ( $\text{cases}\ x2$ )  $\text{auto}$ 
    next
      case ( $\text{UUact}\ x3$ )
        then show  $?thesis$  using  $*$  by ( $\text{cases}\ x3$ )  $\text{auto}$ 
    next
      case ( $\text{Ract}\ x4$ )
        show  $?thesis$ 
        proof ( $\text{cases}\ x4$ )
          case ( $\text{rMyReview}\ x81\ x82\ x83\ x84$ )
            then show  $?thesis$  using  $*$   $\text{Ract}$  by ( $\text{auto}\ \text{simp}\ \text{add} : \text{getNthReview-def}$ )
          next
            case ( $\text{rReviews}\ x91\ x92\ x93\ x94$ )
              then show  $?thesis$  using  $*$   $\text{Ract}$  by ( $\text{clarsimp} ; \text{metis}\ \text{eqExcPID2-imp2}\ ss1$ )
            next
              case ( $\text{rDecs}\ x101\ x102\ x103\ x104$ )
                then show  $?thesis$  using  $*$   $\text{Ract}$  by ( $\text{clarsimp} ; \text{metis}$ )
              next
                case ( $\text{rFinalDec}\ x131\ x132\ x133\ x134$ )
                  then show  $?thesis$  using  $*$   $\text{Ract}$  by ( $\text{clarsimp} ; \text{metis}\ \text{decs-exit}'\ \text{list.sel}(1)$ )
                 $\text{list.simps}(5)\ \text{neq-Nil-conv}$ 
                qed ( $\text{use}\ * \text{Ract}\ \text{in}\ \text{auto}$ )
              next
                case ( $\text{Lact}\ x5$ )
                  then show  $?thesis$  using  $*$  by ( $\text{cases}\ x5 ; \text{auto} ; \text{presburger}$ )
                qed
              qed
            qed
          qed
        qed
      qed
    qed
  qed

lemma  $\text{eqExcPID-step-eqExcPID2}$ :
assumes  $a : a = \text{UUact}\ (\text{uuDec}\ cid\ uid\ p\ PID\ dec)$ 
and  $ss1 : \text{eqExcPID}\ s\ s1$ 
and  $\text{step} : \text{step}\ s\ a = (\text{outOK}, s')$  and  $\text{step1} : \text{step}\ s1\ a = (\text{outOK}, s1')$ 
and  $s : \text{reach}\ s\ \text{reach}\ s1$  and  $PID : PID \in \in \text{paperIDs}\ s\ cid$  and  $\text{ph} : \text{phase}\ s\ cid < \text{notifPH}$ 

```

**shows**  $eqExcPID2\ s'\ s1'$   
**proof** –  
 have  $eqExcPID\ s'\ s1'$  **using**  $assms$  **by** ( $metis\ eqExcPID\ step$ )  
 moreover have  $hd\ (decsPaper\ (paper\ s'\ PID)) = hd\ (decsPaper\ (paper\ s1'\ PID))$   
**using**  $step\ step1$  **unfolding**  $a$   
**apply**( $simp\ add:\ uu\ defs$ )  
**by** ( $metis\ decsPaper.\ simps\ fun\ upd\ same\ list.\ sel(1)\ select\ convs(8)\ surjective\ update\ convs(8)$ )  
**ultimately show**  $?thesis$   
**unfolding**  $eqExcPID\ def\ eqExcPID2\ def\ eqExcPID\ def\ eqExcPID2\ def\ eqExcD2\ eqExcD$  **by**  $auto$   
**qed**

**lemma**  $eqExcPID\ step\ \varphi\ eqExcPID2$ :  
**assumes**  $ss1:\ eqExcPID\ s\ s1$   
**and**  $step:\ step\ s\ a = (ou, s')$  **and**  $step1:\ step\ s1\ a = (ou1, s1')$   
**and**  $\varphi:\ \varphi\ (Trans\ s\ a\ ou\ s')$   
**and**  $s:\ reach\ s\ reach\ s1$  **and**  $PID:\ PID \in\in\ paperIDs\ s\ cid$  **and**  $ph:\ phase\ s\ cid \leq\ disPH$   
**shows**  $eqExcPID2\ s'\ s1'$   
**proof** –  
**obtain**  $cid1\ uid\ p\ dec$  **where**  $a:\ a = UUact\ (uuDec\ cid1\ uid\ p\ PID\ dec)$  **and**  $ou:\ ou = outOK$   
**using**  $\varphi$  **unfolding**  $\varphi\ def2$  **by**  $auto$   
**have**  $PID1:\ PID \in\in\ paperIDs\ s\ cid1$  **using**  $step$  **unfolding**  $a\ ou$  **by** ( $auto\ simp:\ uu\ defs$ )  
**hence**  $cid1:\ cid1 = cid$  **using**  $paperIDs\ equals[OF\ s(1)\ PID]$  **by**  $auto$   
**have**  $\varphi1:\ \varphi\ (Trans\ s1\ a\ ou1\ s1')$  **using**  $\varphi\ ss1$  **by** ( $metis\ eqExcPID\ step\ \varphi\ imp\ step\ step1$ )  
**hence**  $ou1:\ ou1 = outOK$  **using**  $\varphi$  **unfolding**  $\varphi\ def2$  **by**  $auto$   
**show**  $?thesis$   
**using**  $eqExcPID\ step\ eqExcPID2[OF\ a\ ss1\ step[unfolded\ ou]\ step1[unfolded\ ou1]\ s]$   
 $PID\ ph$  **unfolding**  $cid1$  **by**  $auto$   
**qed**

**definition**  $\Delta1 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta1\ s\ vl\ s1\ vl1 \equiv$   
 $(\forall\ cid.\ PID \in\in\ paperIDs\ s\ cid \longrightarrow phase\ s\ cid < disPH) \wedge$   
 $s = s1 \wedge B\ vl\ vl1$

**definition**  $\Delta2 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta2\ s\ vl\ s1\ vl1 \equiv$   
 $\exists\ cid\ uid.$   
 $PID \in\in\ paperIDs\ s\ cid \wedge phase\ s\ cid = disPH \wedge$   
 $isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict \wedge$   
 $eqExcPID\ s\ s1 \wedge B\ vl\ vl1$

**definition**  $\Delta 3 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 3\ s\ vl\ s1\ vl1 \equiv$   
 $\exists\ cid.$   
 $PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid \geq disPH \wedge$   
 $decsPaper\ (paper\ s\ PID) \neq [] \wedge decsPaper\ (paper\ s1\ PID) \neq [] \wedge eqExcPID2\ s$   
 $s1 \wedge$   
 $vl = [] \wedge vl1 = []$

**definition**  $\Delta e :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta e\ s\ vl\ s1\ vl1 \equiv$   
 $vl \neq [] \wedge$   
 $((\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > disPH)$   
 $\vee$   
 $(\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid \geq disPH \wedge$   
 $\neg (\exists\ uid.\ isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict))$   
 $)$

**lemma** *istate- $\Delta 1$* :  
**assumes**  $B: B\ vl\ vl1$   
**shows**  $\Delta 1\ istate\ vl\ istate\ vl1$   
**using**  $B$  **unfolding**  $\Delta 1$ -def  $B$ -def *istate-def* **by** *auto*

**lemma** *unwind-cont- $\Delta 1$* : *unwind-cont*  $\Delta 1\ \{\Delta 1, \Delta 2, \Delta e\}$

**proof**(*rule, simp*)

**let**  $?\Delta = \lambda s\ vl\ s1\ vl1.\ \Delta 1\ s\ vl\ s1\ vl1 \vee \Delta 2\ s\ vl\ s1\ vl1 \vee \Delta e\ s\ vl\ s1\ vl1$

**fix**  $s\ s1 :: state$  **and**  $vl\ vl1 :: value\ list$

**assume**  $rsT: reachNT\ s$  **and**  $rs1: reach\ s1$  **and**  $\Delta 1\ s\ vl\ s1\ vl1$

**hence**  $rs: reach\ s$  **and**  $ss1: s1 = s$

**and**  $vl: vl \neq []$  **and**  $vl1: vl1 \neq []$  **and**  $vl-vl1: last\ vl1 = last\ vl$

**and**  $ph-PID: \bigwedge cid.\ PID \in \in paperIDs\ s\ cid \implies phase\ s\ cid < disPH$

**using** *reachNT-reach* **unfolding**  $\Delta 1$ -def  $B$ -def **by** *auto*

**note**  $vlvl1 = vl\ vl1\ vl-vl1$

**show** *iaction*  $?\Delta\ s\ vl\ s1\ vl1 \vee$

$((vl = [] \longrightarrow vl1 = []) \wedge reaction\ ?\Delta\ s\ vl\ s1\ vl1)$  (**is**  $?iact \vee (- \wedge ?react)$ )

**proof**–

**have**  $?react$  **proof**

**fix**  $a :: act$  **and**  $ou :: out$  **and**  $s' :: state$  **and**  $vl'$

**let**  $?trn = Trans\ s\ a\ ou\ s'$  **let**  $?trn1 = Trans\ s1\ a\ ou\ s'$

**assume**  $step: step\ s\ a = (ou, s')$  **and**  $T: \neg T\ ?trn$  **and**  $c: consume\ ?trn\ vl\ vl'$

**have**  $\varphi: \neg \varphi\ ?trn$

**proof** (*cases a*)

**case** (*UUact x3*)

**then show**  $?thesis$

**using** *step ph-PID*

**by** (*cases x3; fastforce simp: uu-defs*)

**qed** *auto*

```

hence  $vl'$ :  $vl' = vl$  using  $c$  unfolding  $consume-def$  by  $auto$ 
show  $match ?\Delta s s1 vl1 a ou s' vl' \vee ignore ?\Delta s s1 vl1 a ou s' vl'$  (is  $?match$ 
 $\vee ?ignore$ )
proof–
  have  $?match$  proof
    show  $validTrans ?trn1$  unfolding  $ss1$  using  $step$  by  $simp$ 
  next
    show  $consume ?trn1 vl1 vl1$  unfolding  $consume-def ss1$  using  $\varphi$  by  $auto$ 
  next
    show  $\gamma ?trn = \gamma ?trn1$  unfolding  $ss1$  by  $simp$ 
  next
    assume  $\gamma ?trn$  thus  $g ?trn = g ?trn1$  unfolding  $ss1$  by  $simp$ 
  next
    show  $?\Delta s' vl' s' vl1$ 
    proof( $cases \exists cid. PID \in \in paperIDs s cid$ )
      case  $False$  note  $PID = False$ 
      have  $ph-PID'$ :  $\bigwedge cid. PID \in \in paperIDs s' cid \implies phase s' cid < disPH$ 
using  $PID step rs$ 
      apply( $cases a$ )
        subgoal for  $x1$  apply( $cases x1$ ) apply( $fastforce simp: c-defs$ ) $+$  .
        subgoal for  $x2$  apply( $cases x2$ ) apply( $fastforce simp: u-defs$ ) $+$  .
        subgoal for  $x3$  apply( $cases x3$ ) apply( $fastforce simp: uu-defs$ ) $+$  .
        by  $auto$ 
      hence  $\Delta 1 s' vl' s' vl1$  unfolding  $\Delta 1-def B-def vl'$  using  $ph-PID' vlvl1$ 
by  $auto$ 
      thus  $?thesis$  by  $auto$ 
    next
      case  $True$ 
      then obtain  $CID$  where  $PID: PID \in \in paperIDs s CID$  by  $auto$ 
      hence  $ph: phase s CID < disPH$  using  $ph-PID$  by  $auto$ 
      have  $PID'$ :  $PID \in \in paperIDs s' CID$  by ( $metis PID paperIDs-mono$ 
 $step$ )
      show  $?thesis$ 
      proof( $cases phase s' CID < disPH$ )
        case  $True$  note  $ph' = True$ 
        hence  $\Delta 1 s' vl' s' vl1$  unfolding  $\Delta 1-def B-def vl'$  using  $vlvl1 ph' PID'$ 
          by ( $auto; metis reach-PairI paperIDs-equals rs step$ )
          thus  $?thesis$  by  $auto$ 
        next
          case  $False$  note  $ph' = False$ 
          hence  $ph': phase s' CID = disPH$  using  $ph PID step rs$ 
          apply( $cases a$ )
            subgoal for  $x1$  apply( $cases x1$ ) apply( $fastforce simp: c-defs$ ) $+$  .
            subgoal for  $x2$  apply( $cases x2$ ) apply( $fastforce simp: u-defs$ ) $+$  .
            subgoal for  $x3$  apply( $cases x3$ ) apply( $fastforce simp: uu-defs$ ) $+$  .
            by  $auto$ 
          show  $?thesis$ 
          proof( $cases \exists uid. isChair s' CID uid \wedge pref s' uid PID \neq Conflict$ )
            case  $True$ 

```

hence  $\Delta 2$   $s' vl' s' vl1$  **unfolding**  $\Delta 2$ -def  $B$ -def  $vl'$  **using**  $vlvl1$   $ph'$   
*PID'* by *auto*  
 thus *?thesis* by *auto*  
 next  
 case *False*  
 hence  $\Delta e$   $s' vl' s' vl1$  **unfolding**  $\Delta e$ -def  $vl'$  **using**  $vlvl1$   $ph'$  *PID'* by  
*auto*  
 thus *?thesis* by *auto*  
 qed  
 qed  
 qed  
 qed  
 thus *?thesis* by *simp*  
 qed  
 qed  
 thus *?thesis* using  $vl$  by *auto*  
 qed  
 qed

**lemma** *unwind-cont- $\Delta 2$* : *unwind-cont*  $\Delta 2$   $\{\Delta 2, \Delta 3, \Delta e\}$

**proof**(*rule, simp*)

let  $?\Delta = \lambda s vl s1 vl1. \Delta 2 s vl s1 vl1 \vee \Delta 3 s vl s1 vl1 \vee \Delta e s vl s1 vl1$   
**fix**  $s s1 :: state$  **and**  $vl vl1 :: value list$   
**assume**  $rsT$ : *reachNT*  $s$  **and**  $rs1$ : *reach*  $s1$  **and**  $\Delta 2 s vl s1 vl1$   
**then obtain**  $CID uid$  **where**  $uid$ : *isChair*  $s CID uid$  *pref*  $s uid PID \neq Conflict$   
**and**  $rs$ : *reach*  $s$  **and**  $ph$ : *phase*  $s CID = disPH$  (**is**  $ph = -$ )  
**and**  $PID$ :  $PID \in \in paperIDs s CID$  **and**  $ss1$ : *eqExcPID*  $s s1$   
**and**  $vl$ :  $vl \neq []$  **and**  $vl1$ :  $vl1 \neq []$  **and**  $vl-vl1$ : *last*  $vl = last vl1$   
**using** *reachNT-reach* **unfolding**  $\Delta 2$ -def  $B$ -def by *auto*  
**note**  $vlvl1 = vl vl1 vl-vl1$   
**from**  $vl vl1$  **obtain**  $v vl' v1 vl1'$  **where**  $vl$ :  $vl = v \# vl'$  **and**  $vl1$ :  $vl1 = v1 \# vl1'$  by (*metis list.exhaust*)  
**have**  $uid-notin$ :  $uid \notin UIDs$   
**using**  $ph uid reachNT-non-isPC-isChair[OF rsT] PID$  by *fastforce*  
**show**  $iact$   $?\Delta s vl s1 vl1 \vee$   
 $((vl = [] \longrightarrow vl1 = []) \wedge reaction ?\Delta s vl s1 vl1)$  (**is**  $?iact \vee (- \wedge ?react)$ )  
**proof**(*cases*  $vl1' = []$ )  
 case *False* **note**  $vl1' = False$   
 hence  $vl-vl1'$ : *last*  $vl = last vl1'$  **using**  $vl-vl1$  **unfolding**  $vl1$  by *simp*  
 have  $uid1$ : *isChair*  $s1 CID uid$  *pref*  $s1 uid PID \neq Conflict$  **using**  $ss1 uid$   
**unfolding** *eqExcPID-def* by *auto*  
**define**  $a1$  **where**  $a1 \equiv UUact (uuDec CID uid (pass s uid) PID v1)$   
**obtain**  $s1' ou1$  **where**  $step1$ : *step*  $s1 a1 = (ou1, s1')$  **by** (*metis prod.exhaust*)  
**let**  $?trn1 = Trans s1 a1 ou1 s1'$   
**have**  $s1s1'$ : *eqExcPID*  $s1 s1'$  **using**  $a1$ -def  $step1 UUact-uuDec-step-eqExcPID$   
**by** *auto*  
**have**  $ss1'$ : *eqExcPID*  $s s1'$  **using** *eqExcPID-trans*[*OF*  $ss1 s1s1'$ ].  
**hence**  $many-s1'$ :  $PID \in \in paperIDs s1' CID$  *isChair*  $s1' CID uid$  *pref*  $s1' uid PID \neq Conflict$  *phase*  $s1' CID = disPH$

```

    pass s1' uid = pass s uid
    using uid PID ph unfolding eqExcPID-def by auto
    hence more-s1': uid ∈∈ userIDs s1' CID ∈∈ confIDs s1'
    by (metis paperIDs-confIDs reach-PairI roles-userIDs rs1 step1 many-s1'(1))+
    have f: f ?trn1 = v1 unfolding a1-def by simp
    have rs1': reach s1' using rs1 step1 by (auto intro: reach-PairI)
    have ou1: ou1 = outOK
    using step1 uid1 ph unfolding a1-def by (auto simp add: uu-defs many-s1'
more-s1')
    have ?iact proof
      show step s1 a1 = (ou1,s1') by fact
    next
      show φ: φ ?trn1 using ou1 unfolding a1-def by simp
      thus consume ?trn1 vl1 vl1' using f unfolding consume-def vl1 by simp
    next
      show ¬ γ ?trn1 by (simp add: a1-def uid-notin)
    next
      have Δ2 s vl s1' vl1' unfolding Δ2-def B-def using ph PID ss1' uid vl-vl1'
vl1' vl by auto
      thus ?Δ s vl s1' vl1' by simp
    qed
    thus ?thesis by auto
  next
    case True hence vl1: vl1 = [v1] unfolding vl1 by simp
    have ?react proof
      fix a :: act and ou :: out and s' :: state and vll'
      let ?trn = Trans s a ou s'
      let ?ph' = phase s' CID
      assume step: step s a = (ou, s') and T: ¬ T ?trn and c: consume ?trn vl
vll'
      have PID': PID ∈∈ paperIDs s' CID using PID rs by (metis paperIDs-mono
step)
      have uid': isChair s' CID uid ∧ pref s' uid PID ≠ Conflict
      using uid step rs ph PID pref-Conflict-disPH isChair-persistent by auto
      show match ?Δ s s1 vl1 a ou s' vll' ∨ ignore ?Δ s s1 vl1 a ou s' vll' (is
?match ∨ ?ignore)
      proof(cases φ ?trn)
        case False note φ = False
        have vll': vll' = vl using c φ unfolding consume-def by (cases vl) auto
        obtain ou1 and s1' where step1: step s1 a = (ou1,s1') by (metis
prod.exhaust)
        let ?trn1 = Trans s1 a ou1 s1'
        have s's1': eqExcPID s' s1' using eqExcPID-step[OF ss1 step] step1 rs rs1
PID ph by auto
        have φ1: ¬ φ ?trn1 using φ unfolding eqExcPID-step-φ[OF ss1 step step1]
.
      have ?match proof
        show validTrans ?trn1 using step1 by simp
      next

```

```

    show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi$ 1 by auto
  next
  show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
  next
  assume  $\gamma$  ?trn thus g ?trn = g ?trn1
  using eqExcPID-step-out[OF ss1 step step1 rsT rs1 PID ph] by simp

  next
  show ? $\Delta$  s' vll' s1' vl1
  proof(cases ?ph' = disPH)
    case True
    hence  $\Delta$ 2 s' vll' s1' vl1 using PID' s's1' uid' vlvl1 unfolding  $\Delta$ 2-def
  B-def vll' by auto
    thus ?thesis by auto
  next
    case False hence ?ph' > disPH using ph rs step by (metis le-less
  phase-increases)
    hence  $\Delta$ e s' vll' s1' vl1 unfolding  $\Delta$ e-def vll' using vlvl1 PID' by auto
    thus ?thesis by auto
  qed
  qed
  thus ?thesis by simp
  next
  case True note  $\varphi$  = True
  hence vll': vll' = vl' using c unfolding vl consume-def by simp
  obtain cid uid p where a: a = UUact (uuDec cid uid p PID v) and ou: ou
  = outOK
  using  $\varphi$  c PID unfolding vl consume-def  $\varphi$ -def2 vll' by fastforce

  hence  $\gamma$ :  $\neg \gamma$  ?trn using step T rsT by (metis T- $\varphi$ - $\gamma$  True)
  hence f: f ?trn = v using c  $\varphi$  unfolding consume-def vl by auto
  have s's: eqExcPID s' s using eqExcPID-sym[OF  $\varphi$ -step-eqExcPID[OF  $\varphi$ 
  step]] .
  have s's1: eqExcPID s' s1 using eqExcPID-trans[OF s's ss1] .
  have ph': phase s' CID = disPH using s's ph unfolding eqExcPID-def by
  auto
  show ?thesis
  proof(cases vl' = [])
    case False note vl' = False
    hence vl'-vl1: last vl' = last vl1 using vl-vl1 unfolding vl by auto
    have ?ignore proof
      show  $\neg \gamma$  ?trn by fact
    next
      show ? $\Delta$  s' vll' s1 vl1
      proof(cases ?ph' = disPH)
        case True
        hence  $\Delta$ 2 s' vll' s1 vl1 using s's1 PID' uid' vl' vl1 vl-vl1 unfolding
   $\Delta$ 2-def B-def vl vll' by auto
        thus ?thesis by auto
      
```

```

      next
        case False hence  $?ph' > disPH$  using ph rs step by (metis le-less phase-increases)
        hence  $\Delta e\ s'\ vll'\ s1\ vl1$  unfolding  $\Delta e\text{-def}\ vll'$  using vlvl1 vl' PID' by auto
        thus ?thesis by auto
      qed
    qed
  thus ?thesis by auto
next
case True note  $vl' = True$  hence  $vl: vl = [v]$  unfolding vl by simp

  hence  $v1v: v1 = v$  using vl-vl1 unfolding vl1 by simp
  obtain  $s1'\ ou1$  where step1: step s1 a = (ou1, s1') by (metis prod.exhaust)
  let  $?trn1 = Trans\ s1\ a\ ou1\ s1'$ 
  have  $\varphi1: \varphi\ ?trn1$  using eqExcPID-step- $\varphi$ -imp[OF ss1 step step1  $\varphi$ ].
  hence  $ou1: ou1 = outOK$  unfolding  $\varphi$ -def2 by auto
  have ?match proof
  show validTrans ?trn1 using step1 by simp
next
  show consume ?trn1 vl1 [] unfolding consume-def using  $\varphi1$  by (simp add: a vl1 v1v)
next
  show  $\gamma\ ?trn = \gamma\ ?trn1$  unfolding ss1 by simp
next
  assume  $\gamma\ ?trn$  thus  $g\ ?trn = g\ ?trn1$ 
  using eqExcPID-step-out[OF ss1 step step1 rsT rs1 PID ph] by simp
next
  have  $\Delta3\ s'\ vll'\ s1'$  [] unfolding vll' vl'  $\Delta3$ -def
  using PID' ph' eqExcPID-step- $\varphi$ -eqExcPID2[OF ss1 step step1  $\varphi$  rs rs1 PID] ph
  using step step1 unfolding a by (auto simp: uu-defs ou ou1)
  thus  $? \Delta\ s'\ vll'\ s1'$  [] by simp
  qed
  thus ?thesis by simp
  qed
  qed
  qed
  thus ?thesis using vl by auto
  qed
  qed

```

lemma *unwind-cont- $\Delta3$ : unwind-cont  $\Delta3\ \{\Delta3, \Delta e\}$*   
proof(*rule, simp*)  
 let  $? \Delta = \lambda s\ vl\ s1\ vl1. \Delta3\ s\ vl\ s1\ vl1 \vee \Delta e\ s\ vl\ s1\ vl1$   
 fix  $s\ s1 :: state$  and  $vl\ vl1 :: value\ list$   
 assume  $rsT: reachNT\ s$  and  $rs1: reach\ s1$  and  $\Delta3\ s\ vl\ s1\ vl1$   
 then obtain *CID* where  $rs: reach\ s$  and  $ph: phase\ s\ CID \geq disPH$  (*is ?ph  $\geq$*   
-)

```

and PID: PID ∈ ∈ paperIDs s CID
and decs: decsPaper (paper s PID) ≠ [] decsPaper (paper s1 PID) ≠ []
and ss1: eqExcPID2 s s1 and vl: vl = [] and vl1: vl1 = []
using reachNT-reach unfolding  $\Delta 3$ -def by auto
show iaction ? $\Delta$  s vl s1 vl1  $\vee$ 
  ((vl = []  $\rightarrow$  vl1 = [])  $\wedge$  reaction ? $\Delta$  s vl s1 vl1) (is ?iact  $\vee$  ( $\neg$   $\wedge$  ?react))
proof-
  have ?react
  proof
    fix a :: act and ou :: out and s' :: state and vl'
    let ?trn = Trans s a ou s'
    let ?ph' = phase s' CID
    assume step: step s a = (ou, s') and T:  $\neg$  T ?trn and c: consume ?trn vl vl'
    have PID': PID ∈ ∈ paperIDs s' CID using PID rs by (metis paperIDs-mono
step)
    have ?ph': phase s' CID  $\geq$  disPH using ph rs step by (meson dual-order.trans
phase-increases)
    show match ? $\Delta$  s s1 vl1 a ou s' vl'  $\vee$  ignore ? $\Delta$  s s1 vl1 a ou s' vl' (is ?match
 $\vee$  ?ignore)
    proof-
      have  $\varphi$ :  $\neg$   $\varphi$  ?trn and vl': vl' = [] using c unfolding consume-def vl by
auto
      obtain ou1 and s1' where step1: step s1 a = (ou1, s1') by (metis
prod.exhaust)
      let ?trn1 = Trans s1 a ou1 s1'
      have s's1': eqExcPID2 s' s1' using eqExcPID2-step[OF ss1 step step1] .
      have  $\varphi 1$ :  $\neg$   $\varphi$  ?trn1 using  $\varphi$  unfolding eqExcPID2-step- $\varphi$ [OF ss1 step
step1] .
      have PID1: PID ∈ ∈ paperIDs s1 CID using PID ss1 unfolding eqEx-
cPID2-def by auto
      have decs': decsPaper (paper s' PID) ≠ [] decsPaper (paper s1' PID) ≠ []
      by (metis PID PID1 decs nonempty-decsPaper-persist rs step rs1 step1)+
      have ?match proof
        show validTrans ?trn1 using step1 by simp
      next
        show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi 1$  by auto
      next
        show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
      next
        assume  $\gamma$  ?trn thus g ?trn = g ?trn1
        using eqExcPID2-step-out[OF ss1 step step1 rsT rs1 PID ph - decs] by
simp
      next
        have  $\Delta 3$  s' vl' s1' vl1 using ?ph' PID' s's1' unfolding  $\Delta 3$ -def vl1 vl' by
(auto simp: decs')
        thus ? $\Delta$  s' vl' s1' vl1 by simp
      qed
      thus ?thesis by simp
    qed

```

qed  
 thus ?thesis using vl1 by simp  
 qed  
 qed

**definition** *K1exit* where

$K1exit\ cid\ s \equiv$   
 $phase\ s\ cid \geq\ disPH \wedge PID \in\in\ paperIDs\ s\ cid \wedge \neg (\exists\ uid.\ isChair\ s\ cid\ uid \wedge$   
 $pref\ s\ uid\ PID \neq\ Conflict)$

**lemma** *invarNT-K1exit*: *invarNT* (*K1exit* *cid*)

**unfolding** *invarNT-def* **apply** (*safe dest!*: *reachNT-reach*)  
**subgoal** for - **apply**(*cases* *a*)  
**subgoal** for *x1* **apply**(*cases* *x1*)  
**apply** (*fastforce simp add: c-defs K1exit-def geq-noPH-confIDs*) $+$  .  
**subgoal** for *x2* **apply**(*cases* *x2*)  
**apply** (*fastforce simp add: u-defs K1exit-def paperIDs-equals*)  
**apply** (*fastforce simp add: u-defs K1exit-def paperIDs-equals*)  
**apply** (*fastforce simp add: u-defs K1exit-def paperIDs-equals*)  
**apply** (*fastforce simp add: u-defs K1exit-def paperIDs-equals*)  
**apply** (*fastforce simp add: u-defs K1exit-def paperIDs-equals*)  
**subgoal** for *x61* **apply**(*cases* *x61 = cid*)  
**apply** (*fastforce simp add: u-defs K1exit-def paperIDs-equals*) $+$  .  
**apply** (*fastforce simp add: u-defs K1exit-def paperIDs-equals*) .  
**subgoal** for *x3* **apply**(*cases* *x3*) **apply** (*fastforce simp add: uu-defs K1exit-def*) $+$   
 .  
**apply** (*fastforce simp add: uu-defs K1exit-def*) $+$  .  
 done

**lemma** *noVal-K1exit*: *noVal* (*K1exit* *cid*) *v*

**apply**(*rule noφ-noVal*)  
**unfolding** *noφ-def* **apply** *safe*  
**subgoal** for - **apply**(*cases* *a*)  
**apply** (*fastforce simp add: c-defs K1exit-def*)  
**apply** (*fastforce simp add: c-defs K1exit-def*)  
**subgoal** for *x3* **apply**(*cases* *x3*) **apply** (*auto simp add: uu-defs K1exit-def*)  
**apply** (*metis paperIDs-equals reachNT-reach*) .  
**by** *auto*  
 done

**definition** *K2exit* where

$K2exit\ cid\ s \equiv PID \in\in\ paperIDs\ s\ cid \wedge phase\ s\ cid >\ disPH$

**lemma** *invarNT-K2exit*: *invarNT* (*K2exit* *cid*)

**unfolding** *invarNT-def* **apply** (*safe dest!*: *reachNT-reach*)  
**subgoal** for - **apply**(*cases* *a*)  
**subgoal** for *x1* **apply**(*cases* *x1*)  
**apply** (*fastforce simp add: c-defs K2exit-def geq-noPH-confIDs*) $+$  .

```

    subgoal for x2 apply(cases x2)
      apply (fastforce simp add: u-defs K2exit-def paperIDs-equals)+ .
    subgoal for x3 apply(cases x3) apply (fastforce simp add: uu-defs K2exit-def)+
    .
    apply (fastforce simp add: uu-defs K2exit-def)+ .
done

```

```

lemma noVal-K2exit: noVal (K2exit cid) v
apply(rule noφ-noVal)
unfolding noφ-def apply safe
  subgoal for - a apply(cases a)
    apply (fastforce simp add: c-defs K2exit-def)
    apply (fastforce simp add: c-defs K2exit-def)
    subgoal for x3 apply(cases x3) apply (auto simp add: uu-defs K2exit-def)
    using paperIDs-equals reachNT-reach apply fastforce .
  by auto
done

```

```

lemma unwind-exit-Δe: unwind-exit Δe
proof
  fix s s1 :: state and vl vl1 :: value list
  assume rsT: reachNT s and rs1: reach s1 and Δe: Δe s vl s1 vl1
  hence vl: vl ≠ [] using reachNT-reach unfolding Δe-def by auto
  then obtain CID where K1exit CID s ∨ K2exit CID s using Δe unfolding
  K1exit-def K2exit-def Δe-def by auto
  thus vl ≠ [] ∧ exit s (hd vl) apply(simp add: vl)
  by (metis rsT exitI2 invarNT-K1exit noVal-K1exit invarNT-K2exit noVal-K2exit)
qed

```

```

theorem secure: secure
apply(rule unwind-decomp3-secure[of Δ1 Δ2 Δe Δ3])
using
  istate-Δ1
  unwind-cont-Δ1 unwind-cont-Δ2 unwind-cont-Δ3
  unwind-exit-Δe
by auto

```

```

end
theory Decision-NCPC-Aut
imports ../Observation-Setup Decision-Value-Setup Bounded-Deducibility-Security.Compositional-Reasoning
begin

```

## 8.4 Confidentiality protection from users who are not PC members or authors of the paper

We verify the following property:

A group of users UIDs learn nothing about the various updates of a paper's

decision (save for the non-existence of any update) unless/until one of the following happens:

- a user in UIDs becomes a PC member in the paper's conference having no conflict with that paper and the conference moves to the discussion phase, or
- a user in UIDs becomes a PC member in the paper's conference or an author of the paper, and the conference moves to the notification phase

```
fun  $T :: (state,act,out) trans \Rightarrow bool$  where
 $T (Trans - - ou s') =$ 
 $(\exists uid \in UIDs.$ 
 $(\exists cid. PID \in \in paperIDs s' cid \wedge isPC s' cid uid \wedge$ 
 $pref s' uid PID \neq Conflict \wedge phase s' cid \geq disPH)$ 
 $\vee$ 
 $(\exists cid. PID \in \in paperIDs s' cid \wedge isPC s' cid uid \wedge phase s' cid \geq notifPH)$ 
 $\vee$ 
 $isAUT s' uid PID \wedge (\exists cid. PID \in \in paperIDs s' cid \wedge phase s' cid \geq notifPH)$ 
 $)$ 
```

```
declare  $T.simps [simp del]$ 
```

```
definition  $B :: value list \Rightarrow value list \Rightarrow bool$  where
 $B vl vl1 \equiv vl \neq []$ 
```

```
interpretation  $BD\text{-}Security\text{-}IO$  where
 $istate = istate$  and  $step = step$  and
 $\varphi = \varphi$  and  $f = f$  and  $\gamma = \gamma$  and  $g = g$  and  $T = T$  and  $B = B$ 
done
```

```
lemma  $reachNT\text{-}non\text{-}isPC\text{-}isChair$ :
assumes  $reachNT s$  and  $uid \in UIDs$ 
shows
```

```
 $(PID \in \in paperIDs s cid \wedge isPC s cid uid \wedge phase s cid \geq disPH$ 
 $\longrightarrow pref s uid PID = Conflict \wedge phase s cid < notifPH) \wedge$ 
 $(PID \in \in paperIDs s cid \wedge isChair s cid uid \wedge phase s cid \geq disPH$ 
 $\longrightarrow pref s uid PID = Conflict \wedge phase s cid < notifPH) \wedge$ 
 $(isAut s cid uid PID \longrightarrow phase s cid < notifPH)$ 
```

```
using  $assms$ 
```

```
apply  $induct$ 
```

```
apply  $(auto simp: istate-def) []$ 
```

```
apply  $(intro conjI)$ 
```

```
subgoal for  $trn$  apply  $(cases trn, fastforce simp: T.simps) [] .$ 
```

```
subgoal
```

```
using  $T.simps isChair\text{-}isPC[OF reachNT\text{-}reach]$   $not\text{-}less$ 
```

```
 $reach\text{-}PairI[OF reachNT\text{-}reach]$   $validTrans validTrans\text{-}Trans\text{-}srcOf\text{-}actOf\text{-}tgtOf$ 
```

```
by  $(metis (no\text{-}types, opaque\text{-}lifting) isChair\text{-}isPC)$ 
```

by (metis T.elims(3) reach.Step[OF reachNT-reach] isAUT-def isAut-paperIDs not-le-imp-less tgtOf-simps)

**lemma** T- $\varphi$ - $\gamma$ :

**assumes** 1: reachNT s **and** 2: step s a = (ou,s')  $\varphi$  (Trans s a ou s')  
**shows**  $\neg \gamma$  (Trans s a ou s')  
**using** reachNT-non-isPC-isChair[OF 1] 2 **unfolding** T.simps  $\varphi$ -def2  
**by** (force simp add: uu-defs)

**lemma** eqExcPID-step-out:

**assumes** s's1': eqExcPID s s1  
**and** step: step s a = (ou,s') **and** step1: step s1 a = (ou1,s1')  
**and** sT: reachNT s **and** s1: reach s1  
**and** PID: PID  $\in\in$  paperIDs s cid  
**and** UIDs: userOfA a  $\in$  UIDs  
**shows** ou = ou1

**proof** –

**note** Inv = reachNT-non-isPC-isChair[OF sT UIDs]  
**note** eqs = eqExcPID-imp[OF s's1']  
**note** eqs' = eqExcPID-imp1[OF s's1']  
**note** s = reachNT-reach[OF sT]

**note** simps[simp] = c-defs u-defs uu-defs r-defs l-defs Paper-dest-conv eqExcPID-def eqExcPID-def eqExcD  
**note** \* = step step1 eqs eqs' s s1 PID UIDs paperIDs-equals[OF s] Inv

**show** ?thesis

**proof** (cases a)

**case** (Cact x1)

**then show** ?thesis **using** \* **by** (cases x1) auto

**next**

**case** (Uact x2)

**then show** ?thesis **using** \* **by** (cases x2) auto

**next**

**case** (UUact x3)

**then show** ?thesis **using** \* **by** (cases x3) auto

**next**

**case** (Ract x4)

**show** ?thesis

**proof** (cases x4)

**case** (rMyReview x81 x82 x83 x84)

**then show** ?thesis **using** \* Ract **by** (auto simp add: getNthReview-def)

**next**

**case** (rReviews x91 x92 x93 x94)

**then show** ?thesis **using** \* Ract **by** (clarsimp; metis eqExcPID-imp2 s's1')

**next**

**case** (rDecs x101 x102 x103 x104)

**then show** ?thesis **using** \* Ract **by** (clarsimp; metis)

```

next
  case (rFinalDec x131 x132 x133 x134)
    then show ?thesis using * Ract by (clarsimp; metis Suc-leD Suc-leI
not-less-eq-eq)
  qed (use * Ract in auto)
next
  case (Lact x5)
    then show ?thesis using * by (cases x5; auto; presburger)
  qed
qed

```

**definition**  $\Delta 1 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 1\ s\ vl\ s1\ vl1 \equiv$   
 $(\forall\ cid.\ PID \in \in paperIDs\ s\ cid \longrightarrow phase\ s\ cid < disPH) \wedge s = s1 \wedge B\ vl\ vl1$

**definition**  $\Delta 2 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 2\ s\ vl\ s1\ vl1 \equiv$   
 $\exists\ cid\ uid.$   
 $PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid = disPH \wedge$   
 $isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict \wedge$   
 $eqExcPID\ s\ s1$

**definition**  $\Delta 3 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 3\ s\ vl\ s1\ vl1 \equiv$   
 $\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > disPH \wedge eqExcPID\ s\ s1 \wedge vl1 =$   
 $\square$

**definition**  $\Delta e :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta e\ s\ vl\ s1\ vl1 \equiv$   
 $vl \neq \square \wedge$   
 $((\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > disPH)$   
 $\vee$   
 $(\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid \geq disPH \wedge$   
 $\neg (\exists\ uid.\ isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict))$   
 $)$

**lemma** *istate- $\Delta 1$* :  
**assumes**  $B: B\ vl\ vl1$   
**shows**  $\Delta 1\ istate\ vl\ istate\ vl1$   
**using**  $B$  **unfolding**  $\Delta 1$ -def  $B$ -def *istate-def* **by** *auto*

**lemma** *unwind-cont- $\Delta 1$* : *unwind-cont*  $\Delta 1\ \{\Delta 1, \Delta 2, \Delta e\}$

**proof**(*rule, simp*)  
**let**  $?\Delta = \lambda s\ vl\ s1\ vl1.\ \Delta 1\ s\ vl\ s1\ vl1 \vee \Delta 2\ s\ vl\ s1\ vl1 \vee \Delta e\ s\ vl\ s1\ vl1$   
**fix**  $s\ s1 :: state$  **and**  $vl\ vl1 :: value\ list$   
**assume**  $rsT: reachNT\ s$  **and**  $rs1: reach\ s1$  **and**  $\Delta 1\ s\ vl\ s1\ vl1$   
**hence**  $rs: reach\ s$  **and**  $ss1: s1 = s$  **and**  $vl: vl \neq \square$   
**and**  $PID$ -ph:  $\bigwedge\ cid.\ PID \in \in paperIDs\ s\ cid \implies phase\ s\ cid < disPH$

```

using reachNT-reach unfolding  $\Delta$ 1-def B-def by auto
show iaction ? $\Delta$  s vl s1 vl1  $\vee$ 
  ((vl = []  $\longrightarrow$  vl1 = [])  $\wedge$  reaction ? $\Delta$  s vl s1 vl1) (is ?iact  $\vee$  (-  $\wedge$  ?react))
proof-
  have ?react proof
    fix a :: act and ou :: out and s' :: state and vl'
    let ?trn = Trans s a ou s' let ?trn1 = Trans s1 a ou s'
    assume step: step s a = (ou, s') and T:  $\neg$  T ?trn and c: consume ?trn vl vl'
    have  $\varphi$ :  $\neg$   $\varphi$  ?trn
    proof (cases a)
      case (UUact x3)
      then show ?thesis
        using step PID-ph
        by (cases x3; fastforce simp: uu-defs)
    qed auto
    hence vl': vl' = vl using c unfolding consume-def by auto
    show match ? $\Delta$  s s1 vl1 a ou s' vl'  $\vee$  ignore ? $\Delta$  s s1 vl1 a ou s' vl' (is ?match
 $\vee$  ?ignore)
    proof-
      have ?match proof
        show validTrans ?trn1 unfolding ss1 using step by simp
      next
        show consume ?trn1 vl1 vl1 unfolding consume-def ss1 using  $\varphi$  by auto
      next
        show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
      next
        assume  $\gamma$  ?trn thus g ?trn = g ?trn1 unfolding ss1 by simp
      next
        show ? $\Delta$  s' vl' s' vl1
        proof (cases  $\exists$  cid. PID  $\in$  paperIDs s cid)
          case False note PID = False
          have PID-ph':  $\bigwedge$  cid. PID  $\in$  paperIDs s' cid  $\implies$  phase s' cid < disPH
        using PID step rs
          apply (cases a)
          subgoal for - x1 apply (cases x1) apply (fastforce simp: c-defs)+ .
          subgoal for - x2 apply (cases x2) apply (fastforce simp: u-defs)+ .
          subgoal for - x3 apply (cases x3) apply (fastforce simp: uu-defs)+ .
          by auto
        hence  $\Delta$ 1 s' vl' s' vl1 unfolding  $\Delta$ 1-def B-def vl' using PID-ph' vl by
auto
        thus ?thesis by auto
      next
        case True
        then obtain CID where PID: PID  $\in$  paperIDs s CID by auto
        hence ph: phase s CID < disPH using PID-ph by auto
        have PID': PID  $\in$  paperIDs s' CID by (metis PID paperIDs-mono
step)
        show ?thesis
        proof (cases phase s' CID < disPH)

```

```

      case True note ph' = True
      hence  $\Delta 1$  s' vl' s' vl1 unfolding  $\Delta 1$ -def B-def vl' using vl ph' PID'
apply auto
      by (metis reach-PairI paperIDs-equals rs step)
      thus ?thesis by auto
next
      case False note ph' = False
      hence ph': phase s' CID = disPH using ph PID step rs
      apply(cases a)
      subgoal for x1 apply(cases x1) apply(fastforce simp: c-defs)+ .
      subgoal for x2 apply(cases x2) apply(fastforce simp: u-defs)+ .
      subgoal for x3 apply(cases x3) apply(fastforce simp: uu-defs)+ .
      by auto
      show ?thesis
      proof(cases  $\exists$  uid. isChair s' CID uid  $\wedge$  pref s' uid PID  $\neq$  Conflict)
      case True
      hence  $\Delta 2$  s' vl' s' vl1 unfolding  $\Delta 2$ -def vl' using vl ph' PID' by
auto
      thus ?thesis by auto
next
      case False
      hence  $\Delta e$  s' vl' s' vl1 unfolding  $\Delta e$ -def vl' using vl ph' PID' by
auto
      thus ?thesis by auto
      qed
      qed
      qed
      qed
      thus ?thesis by simp
      qed
      qed
      thus ?thesis using vl by auto
      qed
      qed

```

**lemma** *unwind-cont- $\Delta 2$* : *unwind-cont*  $\Delta 2$  { $\Delta 2, \Delta 3, \Delta e$ }

**proof**(rule, simp)

let  $?\Delta = \lambda s vl s1 vl1. \Delta 2 s vl s1 vl1 \vee \Delta 3 s vl s1 vl1 \vee \Delta e s vl s1 vl1$

fix s s1 :: state and vl vl1 :: value list

assume rsT: reachNT s and rs1: reach s1 and  $\Delta 2$  s vl s1 vl1

then obtain CID uid where uid: isChair s CID uid pref s uid PID  $\neq$  Conflict

and rs: reach s and ph: phase s CID = disPH (is ?ph = -)

and PID: PID  $\in$  paperIDs s CID and ss1: eqExcPID s s1

using reachNT-reach unfolding  $\Delta 2$ -def by auto

hence uid-notin: uid  $\notin$  UIDs using ph reachNT-non-isPC-isChair[OF rsT] by force

show iaction  $?\Delta$  s vl s1 vl1  $\vee$

((vl = []  $\longrightarrow$  vl1 = []))  $\wedge$  reaction  $?\Delta$  s vl s1 vl1 (is ?iact  $\vee$  (-  $\wedge$  ?react))

**proof**(cases vl1)

```

    case (Cons v1 vl1') note vl1 = Cons
      have uid1: isChair s1 CID uid pref s1 uid PID ≠ Conflict using ss1 uid
  unfolding eqExcPID-def by auto
    define a1 where a1 ≡ UUact (uuDec CID uid (pass s uid) PID v1)
    obtain s1' ou1 where step1: step s1 a1 = (ou1, s1') by (metis prod.exhaust)
    let ?trn1 = Trans s1 a1 ou1 s1'
    have s1s1': eqExcPID s1 s1' using a1-def step1 UUact-uuDec-step-eqExcPID
  by auto
    have ss1': eqExcPID s s1' using eqExcPID-trans[OF ss1 s1s1'] .
    hence many-s1': PID ∈∈ paperIDs s1' CID isChair s1' CID uid
      pref s1' uid PID ≠ Conflict phase s1' CID = disPH
      pass s1' uid = pass s uid
    using uid PID ph unfolding eqExcPID-def by auto
    hence more-s1': uid ∈∈ userIDs s1' CID ∈∈ confIDs s1'
  by (metis paperIDs-confIDs reach-PairI roles-userIDs rs1 step1 many-s1'(1))+
    have f: f ?trn1 = v1 unfolding a1-def by simp
    have rs1': reach s1' using rs1 step1 by (auto intro: reach-PairI)
    have ou1: ou1 = outOK
    using step1 uid1 ph unfolding a1-def by (auto simp add: uu-defs many-s1'
  more-s1')
    have ?iact proof
      show step s1 a1 = (ou1, s1') by fact
    next
      show φ: φ ?trn1 using ou1 unfolding a1-def by simp
      thus consume ?trn1 vl1 vl1' using f unfolding consume-def vl1 by simp
    next
      show ¬ γ ?trn1 by (simp add: a1-def uid-notin)
    next
      have Δ2 s vl s1' vl1' unfolding Δ2-def using ph PID ss1' uid by auto
      thus ?Δ s vl s1' vl1' by simp
    qed
    thus ?thesis by auto
  next
  case Nil note vl1 = Nil
  have ?react proof
    fix a :: act and ou :: out and s' :: state and vl'
    let ?trn = Trans s a ou s'
    let ?ph' = phase s' CID
    assume step: step s a = (ou, s') and T: ¬ T ?trn and c: consume ?trn vl vl'
  have PID': PID ∈∈ paperIDs s' CID using PID rs by (metis paperIDs-mono
  step)
    have uid': isChair s' CID uid ∧ pref s' uid PID ≠ Conflict
    using uid step rs ph PID pref-Conflict-disPH isChair-persistent by auto
    show match ?Δ s s1 vl1 a ou s' vl' ∨ ignore ?Δ s s1 vl1 a ou s' vl' (is ?match
  ∨ ?ignore)
    proof (cases φ ?trn)
      case False note φ = False
      have vl: vl' = vl using c φ unfolding consume-def by (cases vl) auto
      obtain ou1 and s1' where step1: step s1 a = (ou1, s1') by (metis

```

```

prod.exhaust)
  let ?trn1 = Trans s1 a ou1 s1'
  have s's1': eqExcPID s' s1' using eqExcPID-step[OF ss1 step step1] .
  have  $\varphi$ 1:  $\neg \varphi$  ?trn1 using  $\varphi$  unfolding eqExcPID-step- $\varphi$ [OF ss1 step step1]
  .
  have ?match proof
    show validTrans ?trn1 using step1 by simp
  next
    show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi$ 1 by auto
  next
    show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
  next
    assume  $\gamma$  ?trn thus  $g$  ?trn =  $g$  ?trn1
    using eqExcPID-step-out[OF ss1 step step1 rsT rs1 PID] by simp
  next
    show ? $\Delta$  s' vl' s1' vl1
    proof(cases ?ph' = disPH)
      case True
        hence  $\Delta$ 2 s' vl' s1' vl1 using PID' s's1' uid' unfolding  $\Delta$ 2-def by
auto
      thus ?thesis by auto
    next
      case False hence ?ph' > disPH
        using ph rs step by (metis le-less phase-increases)
        hence  $\Delta$ 3 s' vl' s1' vl1 using s's1' PID' unfolding  $\Delta$ 3-def vl1 by auto
        thus ?thesis by auto
    qed
  qed
  thus ?thesis by simp
next
case True note  $\varphi$  = True
  have s's: eqExcPID s' s using eqExcPID-sym[OF  $\varphi$ -step-eqExcPID[OF  $\varphi$ 
step]] .
  have s's1: eqExcPID s' s1 using eqExcPID-trans[OF s's ss1] .
  have ?ignore proof
    show  $\neg \gamma$  ?trn using T- $\varphi$ - $\gamma$   $\varphi$  rsT step by auto
  next
    show ? $\Delta$  s' vl' s1 vl1
    proof(cases ?ph' = disPH)
      case True
        hence  $\Delta$ 2 s' vl' s1 vl1 using s's1 PID' uid' unfolding  $\Delta$ 2-def by auto
        thus ?thesis by auto
    next
      case False hence ?ph' > disPH
        using ph rs step by (metis le-less phase-increases)
        hence  $\Delta$ 3 s' vl' s1 vl1 using s's1 PID' unfolding  $\Delta$ 3-def vl1 by auto
        thus ?thesis by auto
    qed
  qed

```

```

    thus ?thesis by auto
  qed
  qed
  thus ?thesis using vl1 by auto
  qed
  qed

lemma unwind-cont- $\Delta\mathcal{B}$ : unwind-cont  $\Delta\mathcal{B}$  { $\Delta\mathcal{B}, \Delta e$ }
proof(rule, simp)
  let ? $\Delta$  =  $\lambda s vl s1 vl1. \Delta\mathcal{B} s vl s1 vl1 \vee \Delta e s vl s1 vl1$ 
  fix s s1 :: state and vl vl1 :: value list
  assume rsT: reachNT s and rs1: reach s1 and  $\Delta\mathcal{B}$ :  $\Delta\mathcal{B} s vl s1 vl1$ 
  then obtain CID where
    rs: reach s and ph: phase s CID > disPH (is ?ph > -) and PID: PID  $\in \in$ 
    paperIDs s CID
  and ss1: eqExcPID s s1 and vl1: vl1 = []
  using reachNT-reach unfolding  $\Delta\mathcal{B}$ -def by auto
  show iaction ? $\Delta$  s vl s1 vl1  $\vee$ 
    ((vl = []  $\longrightarrow$  vl1 = [])  $\wedge$  reaction ? $\Delta$  s vl s1 vl1) (is ?iact  $\vee$  (-  $\wedge$  ?react))
  proof-
    have ?react
    proof
      fix a :: act and ou :: out and s' :: state and vl'
      let ?trn = Trans s a ou s'
      let ?ph' = phase s' CID
      assume step: step s a = (ou, s') and T:  $\neg T$  ?trn and c: consume ?trn vl vl'
      have ph': ?ph' > disPH using ph step by (metis less-le-trans phase-increases)
      have PID': PID  $\in \in$  paperIDs s' CID using PID step by (metis paperIDs-mono)

      show match ? $\Delta$  s s1 vl1 a ou s' vl'  $\vee$  ignore ? $\Delta$  s s1 vl1 a ou s' vl' (is ?match
       $\vee$  ?ignore)
    proof-
      have  $\varphi$ :  $\neg \varphi$  ?trn using ph step unfolding  $\varphi$ -def2 apply (auto simp:
      uu-defs)
      using PID less-not-refl2 paperIDs-equals rs by blast
      have vl: vl' = vl using c  $\varphi$  unfolding consume-def by (cases vl) auto
      obtain ou1 and s1' where step1: step s1 a = (ou1, s1') by (metis
      prod.exhaust)
      let ?trn1 = Trans s1 a ou1 s1'
      have s's1': eqExcPID s' s1' using eqExcPID-step[OF ss1 step step1] .
      have  $\varphi$ 1:  $\neg \varphi$  ?trn1 using  $\varphi$  unfolding eqExcPID-step- $\varphi$ [OF ss1 step step1]
      .

      have ?match proof
        show validTrans ?trn1 using step1 by simp
      next
      show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi$ 1 by auto
      next
      show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
      next
    end
  end
end

```

```

    assume  $\gamma$  ?trn thus  $g$  ?trn =  $g$  ?trn1
    using eqExcPID-step-out[OF ss1 step step1 rsT rs1 PID -] ph by simp
next
    have  $\Delta\exists$   $s'$   $vl'$   $s1'$   $vl1$  using ph' PID'  $s's1'$  unfolding  $\Delta\exists$ -def  $vl1$  by
auto
    thus ? $\Delta$   $s'$   $vl'$   $s1'$   $vl1$  by simp
qed
thus ?thesis by simp
qed
qed
thus ?thesis using  $vl1$  by simp
qed
qed

```

**definition** *K1exit* where

```

K1exit  $cid$   $s$   $\equiv$ 
( $PID \in \in$  paperIDs  $s$   $cid$   $\wedge$  phase  $s$   $cid$   $\geq$  disPH  $\wedge$   $\neg$  ( $\exists$  uid. isChair  $s$   $cid$   $uid$   $\wedge$ 
pref  $s$   $uid$   $PID \neq$  Conflict))

```

**lemma** *invarNT-K1exit*: *invarNT* (*K1exit*  $cid$ )

**unfolding** *invarNT-def* **apply** (*safe dest!*: *reachNT-reach*)

```

subgoal for - a apply(cases  $a$ )
subgoal for  $x1$  apply(cases  $x1$ )
  apply (fastforce simp add: c-defs K1exit-def geq-noPH-confIDs) $+$  .
subgoal for  $x2$  apply(cases  $x2$ )
  apply (fastforce simp add: u-defs K1exit-def paperIDs-equals)
  apply (fastforce simp add: u-defs K1exit-def paperIDs-equals)
  apply (fastforce simp add: u-defs K1exit-def paperIDs-equals)
  apply (fastforce simp add: u-defs K1exit-def paperIDs-equals)
  apply (fastforce simp add: u-defs K1exit-def paperIDs-equals)
subgoal for  $x61$  apply(cases  $x61 = cid$ )
  apply (fastforce simp add: u-defs K1exit-def paperIDs-equals) $+$  .
  apply (fastforce simp add: u-defs K1exit-def paperIDs-equals) .
subgoal for  $x3$  apply(cases  $x3$ ) apply (fastforce simp add: uu-defs K1exit-def) $+$ 
.
  apply (fastforce simp add: u-defs K1exit-def paperIDs-equals) $+$  .
done

```

**lemma** *noVal-K1exit*: *noVal* (*K1exit*  $cid$ )  $v$

**apply**(*rule no $\varphi$ -noVal*)

**unfolding** *no $\varphi$ -def* **apply** *safe*

```

subgoal for - a apply(cases  $a$ )
  apply (fastforce simp add: c-defs K1exit-def)
  apply (fastforce simp add: c-defs K1exit-def)
subgoal for  $x3$  apply(cases  $x3$ ) apply (auto simp add: uu-defs K1exit-def)
  apply(metis paperIDs-equals reachNT-reach) .
by auto
done

```

**definition** *K2exit* where

$K2exit\ cid\ s \equiv PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > disPH$

**lemma** *invarNT-K2exit*: *invarNT* (*K2exit cid*)

**unfolding** *invarNT-def* **apply** (*safe dest!*: *reachNT-reach*)

**subgoal for** - *a* **apply**(*cases a*)

**subgoal for** *x1* **apply**(*cases x1*)

**apply** (*fastforce simp add: c-defs K2exit-def geq-noPH-confIDs*)<sup>+</sup> .

**subgoal for** *x2* **apply**(*cases x2*)

**apply** (*fastforce simp add: u-defs K2exit-def paperIDs-equals*)<sup>+</sup> .

**subgoal for** *x3* **apply**(*cases x3*)

**apply** (*fastforce simp add: uu-defs K2exit-def*)<sup>+</sup> .

**by** *auto*

**done**

**lemma** *noVal-K2exit*: *noVal* (*K2exit cid*) *v*

**apply**(*rule noφ-noVal*)

**unfolding** *noφ-def* **apply** *safe*

**subgoal for** - *a* **apply**(*cases a*)

**apply** (*fastforce simp add: c-defs K2exit-def*)

**apply** (*fastforce simp add: c-defs K2exit-def*)

**subgoal for** *x3* **apply**(*cases x3*)

**apply** (*auto simp add: uu-defs K2exit-def*)

**apply** (*metis less-not-refl paperIDs-equals reachNT-reach*) .

**by** *auto*

**done**

**lemma** *unwind-exit-Δe*: *unwind-exit*  $\Delta e$

**proof**

**fix** *s s1* :: *state* **and** *vl vl1* :: *value list*

**assume** *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and**  $\Delta e$ :  $\Delta e\ s\ vl\ s1\ vl1$

**hence** *vl*:  $vl \neq []$  **using** *reachNT-reach* **unfolding**  $\Delta e$ -*def* **by** *auto*

**then obtain** *CID* **where** *K1exit CID s*  $\vee$  *K2exit CID s* **using**  $\Delta e$  **unfolding**  
*K1exit-def K2exit-def Δe-def* **by** *auto*

**thus**  $vl \neq [] \wedge exit\ s\ (hd\ vl)$  **apply**(*simp add: vl*)

**by** (*metis rsT exitI2 invarNT-K1exit noVal-K1exit invarNT-K2exit noVal-K2exit*)  
**qed**

**theorem** *secure*: *secure*

**apply**(*rule unwind-decomp3-secure*[*of*  $\Delta 1\ \Delta 2\ \Delta e\ \Delta 3$ ])

**using**

*istate-Δ1*

*unwind-cont-Δ1 unwind-cont-Δ2 unwind-cont-Δ3*

*unwind-exit-Δe*

**by** *auto*

**end**

**theory** *Decision-All*

```

imports
  Decision-NCPC
  Decision-NCPC-Aut
begin

end
theory Reviewer-Assignment-Intro
imports ../Safety-Properties
begin

```

## 9 Reviewer Assignment Confidentiality

In this section, we prove confidentiality properties for the assignment of reviewers to a paper PID submitted to a conference.

The secrets (values) of interest are taken to be pairs (uid,Uids), where uid is a user and Uids is a set of users. The pairs arise from actions that appoint reviewers to the paper PID:

- uid is the appointed reviewer
- Uids is the set of PC members having no conflict with the paper

The use of the second component, which turns out to be the same for the entire sequence of values<sup>3</sup> is needed in order to express the piece of information (knowledge) that the appointed reviewers are among the non-conflicted PC members.<sup>4</sup>

Here, we have two points of compromise between the bound and the trigger (which yield two properties). Let

- T1 denote “PC membership having no conflict with that paper and the conference having moved to the reviewing phase”
- T2 denote “authorship of the paper and the conference having moved to the notification phase”

The two trigger-bound combinations are:

- weak trigger (T1 or T2) paired with strong bound (allowing to learn nothing beyond the public knowledge that the reviewers are among PC members having no conflict with that paper)

---

<sup>3</sup>This is because conflicts can no longer be changed at the time when reviewers can be appointed, i.e., in the reviewing phase.

<sup>4</sup>In CoCon, only PC members can be appointed as reviewers; there is no subreviewing facility.

- strong trigger (T1) paired with weak bound (allowing to additionally learn the number of reviewers)

end

**theory** *Reviewer-Assignment-Value-Setup*  
**imports** *Reviewer-Assignment-Intro*  
**begin**

## 9.1 Preliminaries

**declare** *updates-commute-paper*[*simp*]

**consts** *PID* :: *paperID*

**definition** *eqExcRLR* :: *role list*  $\Rightarrow$  *role list*  $\Rightarrow$  *bool* **where**  
*eqExcRLR* *rl rl1*  $\equiv$  [*r*  $\leftarrow$  *rl* .  $\neg$  *isRevRoleFor* *PID* *r*] = [*r*  $\leftarrow$  *rl1* .  $\neg$  *isRevRoleFor* *PID* *r*]

**lemma** *eqExcRLR-set*:

**assumes** 1: *eqExcRLR* *rl rl1* **and** 2:  $\neg$  *isRevRoleFor* *PID* *r*

**shows**  $r \in \in rl \iff r \in \in rl1$

**proof** –

**have**  $set ([r \leftarrow rl . \neg isRevRoleFor PID r]) = set ([r \leftarrow rl1 . \neg isRevRoleFor PID r])$

**using** 1 **unfolding** *eqExcRLR-def* **by** *auto*

**thus** *?thesis* **using** 2 **unfolding** *set-filter* **by** *auto*

**qed**

**lemmas** *eqExcRLR* = *eqExcRLR-def*

**lemma** *eqExcRLR-eq*[*simp,intro!*]: *eqExcRLR* *rl rl*

**unfolding** *eqExcRLR* **by** *auto*

**lemma** *eqExcRLR-sym*:

**assumes** *eqExcRLR* *rl rl1*

**shows** *eqExcRLR* *rl1 rl*

**using** *assms* **unfolding** *eqExcRLR* **by** *auto*

**lemma** *eqExcRLR-trans*:

**assumes** *eqExcRLR* *rl rl1* **and** *eqExcRLR* *rl1 rl2*

**shows** *eqExcRLR* *rl rl2*

**using** *assms* **unfolding** *eqExcRLR* **by** *auto*

**lemma** *eqExcRLR-imp*:

**assumes** *s*: *reach* *s* **and** *pid*: *pid*  $\neq$  *PID* **and**

1: *eqExcRLR* (*roles* *s* *cid* *uid*) (*roles* *s1* *cid* *uid*)

**shows**

```

isRevNth s cid uid pid = isRevNth s1 cid uid pid ∧
isRev s cid uid pid = isRev s1 cid uid pid ∧
getRevRole s cid uid pid = getRevRole s1 cid uid pid ∧
getReviewIndex s cid uid pid = getReviewIndex s1 cid uid pid (is ?A ∧ ?B ∧ ?C
∧ ?D)
proof(intro conjI)
  show A: ?A
    apply(rule ext)
    using 1 by (metis eqExcRLR-set isRevRoleFor.simps(1) pid)
  show B: ?B using A unfolding isRev-def2 by auto
  show C: ?C
    apply(cases isRev s cid uid pid)
    subgoal by (metis A B getRevRole-Some-Rev-isRevNth isRevNth-equals is-
Rev-getRevRole2 s)
    by (metis B Bex-set-list-ex find-None-iff getRevRole-def isRev-def)
  show D: ?D unfolding getReviewIndex-def using C by auto
qed

```

```

lemma eqExcRLR-imp2:
assumes eqExcRLR (roles s cid uid) (roles s1 cid uid)
shows
isPC s cid uid = isPC s1 cid uid ∧
isChair s cid uid = isChair s1 cid uid ∧
isAut s cid uid = isAut s1 cid uid
by (metis (opaque-lifting, no-types) assms eqExcRLR-set isRevRoleFor.simps)

```

```

lemma filter-eq-imp:
assumes  $\bigwedge x. P x \implies Q x$ 
and filter Q xs = filter Q ys
shows filter P xs = filter P ys
using assms filter-filter
proof–
  have filter P xs = filter P (filter Q xs)
  unfolding filter-filter using assms by metis
  also have ... = filter P (filter Q ys) using assms by simp
  also have ... = filter P ys unfolding filter-filter using assms by metis
  finally show ?thesis .
qed

```

```

lemma arg-cong3: a = a1  $\implies$  b = b1  $\implies$  c = c1  $\implies$  h a b c = h a1 b1 c1
by auto

```

```

lemmas map-concat-cong1 = arg-cong[where f = concat, OF arg-cong2[where f
= map, OF - refl]]
lemmas If-cong1 = arg-cong3[where h = If, OF - refl refl]

```

```

lemma diff-cong1: a = a1  $\implies$  (a  $\neq$  b) = (a1  $\neq$  b) by auto

```

**lemma** *isRev-pref-notConflict*:  
**assumes** *reach s* **and** *isRev s cid uid pid*  
**shows** *pref s uid pid*  $\neq$  *Conflict*  
**by** (*metis assms pref-Conflict-isRev*)

**lemma** *isRev-pref-notConflict-isPC*:  
**assumes** *reach s* **and** *isRev s cid uid pid*  
**shows** *pref s uid pid*  $\neq$  *Conflict*  $\wedge$  *isPC s cid uid*  
**by** (*metis assms(1) assms(2) isRev-isPC isRev-pref-notConflict*)

**lemma** *eqExcRLLR-imp-isRevRole-imp*:  
**assumes** *eqExcRLLR rl rl1*  
**shows**  $[r \leftarrow rl. \neg \text{isRevRole } r] = [r \leftarrow rl1. \neg \text{isRevRole } r]$   
**using** *assms filter-eq-imp unfolding eqExcRLLR-def*  
**by** (*metis isRevRole.simps(1) isRevRoleFor.elims(2)*)

**lemma** *notIsPC-eqExLRL-roles-eq*:  
**assumes** *s: reach s* **and** *s1: reach s1* **and** *PID: PID  $\in$  paperIDs s cid*  
**and** *pc:  $\neg$  isPC s cid uid*  
**and** *eq: eqExcRLLR (roles s cid uid) (roles (s1::state) cid uid)*  
**shows** *roles s cid uid = roles s1 cid uid*  
**proof** –  
**have**  $\neg \text{isPC } s1 \text{ cid uid}$  **using** *pc eqExcRLLR-imp2[OF eq]* **by** *auto*  
**hence**  $\neg \text{isRev } s \text{ cid uid } PID \wedge \neg \text{isRev } s1 \text{ cid uid } PID$  **using** *pc s s1 PID*  
**by** (*metis isRev-pref-notConflict-isPC*)  
**thus** *?thesis* **using** *eq unfolding eqExcRLLR-def*  
**by** (*metis Bex-set-list-ex filter-id-conv isRev-def*)  
**qed**

**lemma** *foo1*:  $P a \implies [r \leftarrow \text{List.insert } a \ l. P \ r] = (\text{if } a \in \text{set } l \text{ then filter } P \ l \text{ else } a \# \text{filter } P \ l)$   
**by** (*metis filter.simps(2) in-set-insert not-in-set-insert*)

**lemma** *foo2*:  $\llbracket \text{eqExcRLLR } rl \ rl'; \neg \text{isRevRoleFor } PID \ x \rrbracket \implies \text{eqExcRLLR } (\text{List.insert } x \ rl) \ (\text{List.insert } x \ rl')$   
**unfolding** *eqExcRLLR-def*  
**apply** (*auto simp: foo1*)  $\square$   
**apply** (*metis eqExcRLLR-def eqExcRLLR-set isRevRoleFor.simps*) +  
**done**

**lemma** *foo3*:  
**assumes** *eqExcRLLR rl rl' isRevRoleFor PID x*  
**shows** *eqExcRLLR (List.insert x rl) (rl')*  
**and** *eqExcRLLR (rl) (List.insert x rl')*  
**using** *assms*  
**unfolding** *eqExcRLLR-def*  
**by** (*auto simp: List.insert-def*)

The notion of two states being equal everywhere except on the reviewer roles

for PID:

**definition**  $eqExcPID :: state \Rightarrow state \Rightarrow bool$  **where**  
 $eqExcPID\ s\ s1 \equiv$   
 $confIDs\ s = confIDs\ s1 \wedge conf\ s = conf\ s1 \wedge$   
 $userIDs\ s = userIDs\ s1 \wedge pass\ s = pass\ s1 \wedge user\ s = user\ s1$   
 $\wedge$   
 $(\forall\ cid\ uid.\ eqExcRLR\ (roles\ s\ cid\ uid)\ (roles\ s1\ cid\ uid))$   
 $\wedge$   
 $paperIDs\ s = paperIDs\ s1$   
 $\wedge$   
 $paper\ s = paper\ s1$   
 $\wedge$   
 $pref\ s = pref\ s1 \wedge$   
 $voronkov\ s = voronkov\ s1 \wedge$   
 $news\ s = news\ s1 \wedge phase\ s = phase\ s1$

**lemma**  $eqExcPID\text{-}eq[simp,intro!]$ :  $eqExcPID\ s\ s$   
**unfolding**  $eqExcPID\text{-}def$  **by**  $auto$

**lemma**  $eqExcPID\text{-}sym$ :  
**assumes**  $eqExcPID\ s\ s1$  **shows**  $eqExcPID\ s1\ s$   
**using**  $assms\ eqExcRLR\text{-}sym$  **unfolding**  $eqExcPID\text{-}def$  **by**  $auto$

**lemma**  $eqExcPID\text{-}trans$ :  
**assumes**  $eqExcPID\ s\ s1$  **and**  $eqExcPID\ s1\ s2$  **shows**  $eqExcPID\ s\ s2$   
**using**  $assms\ eqExcRLR\text{-}trans$  **unfolding**  $eqExcPID\text{-}def$  **by**  $metis$

**lemma**  $eqExcPID\text{-}imp$ :  
 $eqExcPID\ s\ s1 \implies$   
 $confIDs\ s = confIDs\ s1 \wedge conf\ s = conf\ s1 \wedge$   
 $userIDs\ s = userIDs\ s1 \wedge pass\ s = pass\ s1 \wedge user\ s = user\ s1$   
 $\wedge$   
 $eqExcRLR\ (roles\ s\ cid\ uid)\ (roles\ s1\ cid\ uid)$   
 $\wedge$   
 $paperIDs\ s = paperIDs\ s1$   
 $\wedge$   
 $paper\ s = paper\ s1$   
 $\wedge$   
 $pref\ s = pref\ s1 \wedge$   
 $voronkov\ s = voronkov\ s1 \wedge$   
 $news\ s = news\ s1 \wedge phase\ s = phase\ s1 \wedge$   
 $getAllPaperIDs\ s = getAllPaperIDs\ s1$   
**unfolding**  $eqExcPID\text{-}def\ eqExcRLR\text{-}def\ getAllPaperIDs\text{-}def$  **by**  $auto$

**lemma**  $eqExcPID\text{-}imp'$ :  
**assumes**  $s:\ reach\ s$  **and**  $ss1: eqExcPID\ s\ s1$  **and**  $pid: pid \neq PID \vee PID \neq pid$   
**shows**

$isRev\ s\ cid\ uid\ pid = isRev\ s1\ cid\ uid\ pid \wedge$   
 $getRevRole\ s\ cid\ uid\ pid = getRevRole\ s1\ cid\ uid\ pid \wedge$   
 $getReviewIndex\ s\ cid\ uid\ pid = getReviewIndex\ s1\ cid\ uid\ pid$   
**proof** –  
**have** 1:  $eqExcRLR\ (roles\ s\ cid\ uid)\ (roles\ s1\ cid\ uid)$   
**using**  $eqExcPID-imp[OF\ ss1]$  **by** *auto*  
**show** ?thesis **proof** (intro conjI)  
**show** 3:  $isRev\ s\ cid\ uid\ pid = isRev\ s1\ cid\ uid\ pid$   
**by** (metis 1  $eqExcRLR-imp\ pid\ s$ )  
**show** 4:  $getRevRole\ s\ cid\ uid\ pid = getRevRole\ s1\ cid\ uid\ pid$   
**by** (metis 1  $eqExcRLR-imp\ pid\ s$ )  
**show**  $getReviewIndex\ s\ cid\ uid\ pid = getReviewIndex\ s1\ cid\ uid\ pid$   
**unfolding**  $getReviewIndex-def$  **using** 4 **by** *auto*  
**qed**  
**qed**

**lemma**  $eqExcPID-imp1$ :  
 $eqExcPID\ s\ s1 \implies pid \neq PID \vee PID \neq pid \implies$   
 $getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$   
**unfolding**  $eqExcPID-def\ getNthReview-def$   
**by** *auto*

**lemma**  $eqExcPID-imp2$ :  
**assumes**  $reach\ s$  **and**  $eqExcPID\ s\ s1$  **and**  $pid \neq PID \vee PID \neq pid$   
**shows**  $getReviewersReviews\ s\ cid\ pid = getReviewersReviews\ s1\ cid\ pid$   
**proof** –  
**have**  
 $(\lambda uID. if\ isRev\ s\ cid\ uID\ pid\ then\ [(uID,\ getNthReview\ s\ pid\ (getReviewIndex\ s\ cid\ uID\ pid))] else\ []) =$   
 $(\lambda uID. if\ isRev\ s1\ cid\ uID\ pid\ then\ [(uID,\ getNthReview\ s1\ pid\ (getReviewIndex\ s1\ cid\ uID\ pid))] else\ [])$   
**apply**(rule ext)  
**using** *assms* **using** *assms* **by** (auto simp add:  $eqExcPID-imp'\ eqExcPID-imp1$ )  
**thus** ?thesis **unfolding**  $getReviewersReviews-def$  **using** *assms* **by** (simp add:  $eqExcPID-imp$ )  
**qed**

**lemma**  $eqExcPID-imp3$ :  
 $reach\ s \implies eqExcPID\ s\ s1 \implies pid \neq PID \vee PID \neq pid$   
 $\implies$   
 $getNthReview\ s\ pid = getNthReview\ s1\ pid$   
**unfolding**  $eqExcPID-def$  **apply** *auto*  
**apply** (rule ext) **by** (metis  $getNthReview-def$ )

**lemma**  $eqExcPID-cong[simp,\ intro]$ :  
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!confIDs := uu1))$   
 $(s1\ (\!confIDs := uu2))$   
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!conf := uu1))\ (s1$   
 $(\!conf := uu2))$

$$\begin{aligned}
& \wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s (\!|userIDs := uu1\!|)) \\
& (s1 (\!|userIDs := uu2\!|)) \\
& \wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s (\!|pass := uu1\!|)) (s1 \\
& (\!|pass := uu2\!|)) \\
& \wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s (\!|user := uu1\!|)) (s1 \\
& (\!|user := uu2\!|)) \\
& \wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s (\!|roles := uu1\!|)) (s1 \\
& (\!|roles := uu2\!|)) \\
\\
& \wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s (\!|paperIDs := uu1\!|)) \\
& (s1 (\!|paperIDs := uu2\!|)) \\
& \wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s (\!|paper := uu1\!|)) (s1 \\
& (\!|paper := uu2\!|)) \\
\\
& \wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s (\!|pref := uu1\!|)) (s1 \\
& (\!|pref := uu2\!|)) \\
& \wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s (\!|voronkov := uu1\!|)) \\
& (s1 (\!|voronkov := uu2\!|)) \\
& \wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s (\!|news := uu1\!|)) (s1 \\
& (\!|news := uu2\!|)) \\
& \wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s (\!|phase := uu1\!|)) (s1 \\
& (\!|phase := uu2\!|))
\end{aligned}$$

**unfolding** *eqExcPID-def by auto*

A slightly weaker state equivalence that allows differences in the reviews of paper *PID*. It is used for the confidentiality property that doesn't cover the authors of that paper in the notification phase (when the authors will learn the contents of the reviews).

**fun** *eqExcR* :: *paper*  $\Rightarrow$  *paper*  $\Rightarrow$  *bool* **where**  
*eqExcR* (*Paper name info ct reviews dis decs*)  
 (*Paper name1 info1 ct1 reviews1 dis1 decs1*) =  
 (*name = name1*  $\wedge$  *info = info1*  $\wedge$  *ct = ct1*  $\wedge$  *dis = dis1*  $\wedge$  *decs = decs1*)

**lemma** *eqExcR*:  
*eqExcR pap pap1* =  
 (*titlePaper pap = titlePaper pap1*  $\wedge$  *abstractPaper pap = abstractPaper pap1*  $\wedge$   
*contentPaper pap = contentPaper pap1*  $\wedge$   
*disPaper pap = disPaper pap1*  $\wedge$  *decsPaper pap = decsPaper pap1*)  
**by**(*cases pap, cases pap1, auto*)

**lemma** *eqExcR-eq[simp,intro!]*: *eqExcR pap pap*  
**unfolding** *eqExcR* **by** *auto*

**lemma** *eqExcR-sym*:  
**assumes** *eqExcR pap pap1*  
**shows** *eqExcR pap1 pap*  
**using** *assms unfolding eqExcR* **by** *auto*

**lemma** *eqExcR-trans*:  
**assumes** *eqExcR pap pap1 and eqExcR pap1 pap2*  
**shows** *eqExcR pap pap2*  
**using** *assms unfolding eqExcR by auto*

**definition** *eeqExcPID where*  
*eeqExcPID paps paps1*  $\equiv$   
 $\forall pid. \text{if } pid = PID \text{ then } eqExcR (paps \ pid) (paps1 \ pid) \text{ else } paps \ pid = paps1 \ pid$

**lemma** *eeqExcPID-eeq[simp,intro!]*: *eeqExcPID s s*  
**unfolding** *eeqExcPID-def* **by** *auto*

**lemma** *eeqExcPID-sym*:  
**assumes** *eeqExcPID s s1* **shows** *eeqExcPID s1 s*  
**using** *assms eqExcR-sym unfolding eeqExcPID-def by auto*

**lemma** *eeqExcPID-trans*:  
**assumes** *eeqExcPID s s1 and eeqExcPID s1 s2* **shows** *eeqExcPID s s2*  
**using** *assms eqExcR-trans unfolding eeqExcPID-def by simp blast*

**lemma** *eeqExcPID-imp*:  
*eeqExcPID paps paps1*  $\implies eqExcR (paps \ PID) (paps1 \ PID)$   
 $\llbracket eeqExcPID paps paps1; pid \neq PID \rrbracket \implies paps \ pid = paps1 \ pid$   
**unfolding** *eeqExcPID-def* **by** *auto*

**lemma** *eeqExcPID-cong*:  
**assumes** *eeqExcPID paps paps1*  
**and** *pid = PID*  $\implies eqExcR uu uu1$   
**and** *pid*  $\neq PID \implies uu = uu1$   
**shows** *eeqExcPID (paps (pid := uu)) (paps1 (pid := uu1))*  
**using** *assms unfolding eeqExcPID-def by auto*

**lemma** *eeqExcPID-RDD*:  
*eeqExcPID paps paps1*  $\implies$   
 $titlePaper (paps \ PID) = titlePaper (paps1 \ PID) \wedge$   
 $abstractPaper (paps \ PID) = abstractPaper (paps1 \ PID) \wedge$   
 $contentPaper (paps \ PID) = contentPaper (paps1 \ PID) \wedge$   
 $disPaper (paps \ PID) = disPaper (paps1 \ PID) \wedge$   
 $decsPaper (paps \ PID) = decsPaper (paps1 \ PID)$   
**using** *eeqExcPID-def unfolding eqExcR by auto*

**definition** *eqExcPID2*  $:: state \Rightarrow state \Rightarrow bool$  **where**  
*eqExcPID2 s s1*  $\equiv$   
 $confIDs s = confIDs s1 \wedge conf s = conf s1 \wedge$   
 $userIDs s = userIDs s1 \wedge pass s = pass s1 \wedge user s = user s1$   
 $\wedge$

$(\forall \text{ cid uid. eqExcRLR } (\text{roles } s \text{ cid uid}) (\text{roles } s1 \text{ cid uid}))$   
 $\wedge$   
 $\text{paperIDs } s = \text{paperIDs } s1$   
 $\wedge$   
 $\text{eeqExcPID } (\text{paper } s) (\text{paper } s1)$   
 $\wedge$   
 $\text{pref } s = \text{pref } s1 \wedge$   
 $\text{voronkov } s = \text{voronkov } s1 \wedge$   
 $\text{news } s = \text{news } s1 \wedge \text{phase } s = \text{phase } s1$

**lemma**  $\text{eqExcPID2-}[\text{simp,intro!}]$ :  $\text{eqExcPID2 } s \ s$   
**unfolding**  $\text{eqExcPID2-def}$  **by** *auto*

**lemma**  $\text{eqExcPID2-sym}$ :  
**assumes**  $\text{eqExcPID2 } s \ s1$  **shows**  $\text{eqExcPID2 } s1 \ s$   
**using** *assms*  $\text{eqExcPID-sym}$   $\text{eqExcRLR-sym}$  **unfolding**  $\text{eqExcPID2-def}$  **by** *auto*

**lemma**  $\text{eqExcPID2-trans}$ :  
**assumes**  $\text{eqExcPID2 } s \ s1$  **and**  $\text{eqExcPID2 } s1 \ s2$  **shows**  $\text{eqExcPID2 } s \ s2$   
**using** *assms*  $\text{eqExcPID-trans}$   $\text{eqExcRLR-trans}$  **unfolding**  $\text{eqExcPID2-def}$  **by** *metis*

**lemma**  $\text{eqExcPID2-imp}$ :  
 $\text{eqExcPID2 } s \ s1 \implies$   
 $\text{confIDs } s = \text{confIDs } s1 \wedge \text{conf } s = \text{conf } s1 \wedge$   
 $\text{userIDs } s = \text{userIDs } s1 \wedge \text{pass } s = \text{pass } s1 \wedge \text{user } s = \text{user } s1$   
 $\wedge$   
 $\text{eqExcRLR } (\text{roles } s \ \text{cid } \ \text{uid}) (\text{roles } s1 \ \text{cid } \ \text{uid})$   
 $\wedge$   
 $\text{paperIDs } s = \text{paperIDs } s1$   
 $\wedge$   
 $\text{eeqExcPID } (\text{paper } s) (\text{paper } s1)$   
 $\wedge$   
 $\text{pref } s = \text{pref } s1 \wedge$   
 $\text{voronkov } s = \text{voronkov } s1 \wedge$   
 $\text{news } s = \text{news } s1 \wedge \text{phase } s = \text{phase } s1 \wedge$

$\text{getAllPaperIDs } s = \text{getAllPaperIDs } s1$   
**unfolding**  $\text{eqExcPID2-def}$   $\text{eqExcRLR-def}$   $\text{getAllPaperIDs-def}$  **by** *auto*

**lemma**  $\text{eeqExcPID-imp2}$ :  
**assumes**  $\text{pid: pid} \neq \text{PID}$  **and**  
 $1$ :  $\text{eeqExcPID } (\text{paper } s) (\text{paper } s1)$   
**shows**  
 $\text{reviewsPaper } (\text{paper } s \ \text{pid}) = \text{reviewsPaper } (\text{paper } s1 \ \text{pid})$   
**by** (*metis*  $1$   $\text{eeqExcPID-imp}(2)$   $\text{pid}$ )

**lemma** *eqExcPID2-imp'*:

**assumes** *s*: *reach s* **and** *ss1*: *eqExcPID2 s s1* **and** *pid*: *pid*  $\neq$  *PID*  $\vee$  *PID*  $\neq$  *pid*

**shows**

*isRev s cid uid pid* = *isRev s1 cid uid pid*  $\wedge$   
*getRevRole s cid uid pid* = *getRevRole s1 cid uid pid*  $\wedge$   
*getReviewIndex s cid uid pid* = *getReviewIndex s1 cid uid pid*  $\wedge$   
*reviewsPaper (paper s pid)* = *reviewsPaper (paper s1 pid)*

**proof** –

**have** 1: *eqExcRLR (roles s cid uid) (roles s1 cid uid)*

**and** 2: *eeqExcPID (paper s) (paper s1)*

**using** *eqExcPID2-imp'[OF ss1]* **by** *auto*

**show** *?thesis* **proof** (*intro conjI*)

**show** 3: *isRev s cid uid pid* = *isRev s1 cid uid pid*

**by** (*metis 1 eqExcRLR-imp pid s*)

**show** 4: *getRevRole s cid uid pid* = *getRevRole s1 cid uid pid*

**by** (*metis 1 eqExcRLR-imp pid s*)

**show** *getReviewIndex s cid uid pid* = *getReviewIndex s1 cid uid pid*

**unfolding** *getReviewIndex-def* **using** 4 **by** *auto*

**show** *reviewsPaper (paper s pid)* = *reviewsPaper (paper s1 pid)*

**using** *pid 2 unfolding eeqExcPID-def* **by** *auto*

**qed**

**qed**

**lemma** *eqExcPID2-imp1*:

*eqExcPID2 s s1*  $\implies$  *eqExcR (paper s pid) (paper s1 pid)*

*eqExcPID2 s s1*  $\implies$  *pid*  $\neq$  *PID*  $\vee$  *PID*  $\neq$  *pid*  $\implies$

*paper s pid* = *paper s1 pid*  $\wedge$

*getNthReview s pid n* = *getNthReview s1 pid n*

**unfolding** *eqExcPID2-def eeqExcPID-def getNthReview-def*

**apply** *auto* **by** (*metis eqExcR-eq*)

**lemma** *eqExcPID2-imp2*:

**assumes** *reach s* **and** *eqExcPID2 s s1* **and** *pid*  $\neq$  *PID*  $\vee$  *PID*  $\neq$  *pid*

**shows** *getReviewersReviews s cid pid* = *getReviewersReviews s1 cid pid*

**proof** –

**have**

( $\lambda uID$ . *if isRev s cid uID pid then [(uID, getNthReview s pid (getReviewIndex s cid uID pid))] else []*) =

( $\lambda uID$ . *if isRev s1 cid uID pid then [(uID, getNthReview s1 pid (getReviewIndex s1 cid uID pid))] else []*)

**apply**(*rule ext*)

**using** *assms using assms* **by** (*auto simp add: eqExcPID2-imp' eqExcPID2-imp1*)

**thus** *?thesis* **unfolding** *getReviewersReviews-def* **using** *assms* **by** (*simp add: eqExcPID2-imp*)

**qed**

**lemma** *eqExcPID2-imp3*:

*reach s*  $\implies$  *eqExcPID2 s s1*  $\implies$  *pid*  $\neq$  *PID*  $\vee$  *PID*  $\neq$  *pid*

$\implies$

$getNthReview\ s\ pid = getNthReview\ s1\ pid$   
**unfolding**  $eqExcPID2-def$  **apply**  $auto$   
**apply** ( $rule\ ext$ ) **by** ( $metis\ eqExcPID-imp\ getNthReview-def$ )

**lemma**  $eqExcPID2-RDD$ :

$eqExcPID2\ s\ s1 \implies$   
 $titlePaper\ (paper\ s\ PID) = titlePaper\ (paper\ s1\ PID) \wedge$   
 $abstractPaper\ (paper\ s\ PID) = abstractPaper\ (paper\ s1\ PID) \wedge$   
 $contentPaper\ (paper\ s\ PID) = contentPaper\ (paper\ s1\ PID) \wedge$   
 $disPaper\ (paper\ s\ PID) = disPaper\ (paper\ s1\ PID) \wedge$   
 $decsPaper\ (paper\ s\ PID) = decsPaper\ (paper\ s1\ PID)$   
**using**  $eqExcPID2-imp\ eqExcPID-RDD$  **by**  $auto$

**lemma**  $eqExcPID2-cong[simp, intro]$ :

$\bigwedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|confIDs := uu1|))$   
 $(s1\ (\!|confIDs := uu2|))$   
 $\bigwedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|conf := uu1|))$   
 $(s1\ (\!|conf := uu2|))$

$\bigwedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|userIDs := uu1|))$   
 $(s1\ (\!|userIDs := uu2|))$   
 $\bigwedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|pass := uu1|))$   
 $(s1\ (\!|pass := uu2|))$   
 $\bigwedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|user := uu1|))$   
 $(s1\ (\!|user := uu2|))$   
 $\bigwedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|roles := uu1|))$   
 $(s1\ (\!|roles := uu2|))$

$\bigwedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|paperIDs := uu1|))$   
 $(s1\ (\!|paperIDs := uu2|))$   
 $\bigwedge uu1\ uu2. eqExcPID2\ s\ s1 \implies eqExcPID\ uu1\ uu2 \implies eqExcPID2\ (s\ (\!|paper := uu1|))$   
 $(s1\ (\!|paper := uu2|))$

$\bigwedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|pref := uu1|))$   
 $(s1\ (\!|pref := uu2|))$   
 $\bigwedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|voronkov := uu1|))$   
 $(s1\ (\!|voronkov := uu2|))$   
 $\bigwedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|news := uu1|))$   
 $(s1\ (\!|news := uu2|))$   
 $\bigwedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|phase := uu1|))$   
 $(s1\ (\!|phase := uu2|))$

**unfolding**  $eqExcPID2-def$  **by**  $auto$

**lemma**  $eqExcPID2-Paper$ :

**assumes**  $s's1'$ :  $eqExcPID2\ s\ s1$   
**and**  $paper\ s\ pid = Paper\ title\ abstract\ content\ reviews\ dis\ decs$   
**and**  $paper\ s1\ pid = Paper\ title1\ abstract1\ content1\ reviews1\ dis1\ decs1$   
**shows**  $title = title1 \wedge abstract = abstract1 \wedge content = content1 \wedge dis = dis1 \wedge$

```

decs = decs1
using assms unfolding eqExcPID2-def apply (auto simp: eqExcR eqExcPID-def)
by (metis titlePaper.simps abstractPaper.simps contentPaper.simps disPaper.simps
decsPaper.simps)+

```

```

lemma cReview-step-eqExcPID2:
assumes a:
a = Cact (cReview cid uid p PID uid')
and step s a = (ou, s')
shows eqExcPID2 s s'
using assms unfolding eqExcPID2-def eqExcPID-def eqExcRLR-def
apply (auto simp: c-defs)
unfolding List.insert-def by (smt (verit) filter.simps(2) isRevRoleFor.simps(1))

```

## 9.2 Value Setup

```

type-synonym value = userID × userID set

```

```

fun  $\varphi$  :: (state, act, out) trans ⇒ bool where
 $\varphi$  (Trans - (Cact (cReview cid uid p pid uid')) ou -) =
  (pid = PID ∧ ou = outOK)
|
 $\varphi$  - = False

```

```

fun f :: (state, act, out) trans ⇒ value where
f (Trans s (Cact (cReview cid uid p pid uid')) - s') =
  (uid', {uid'. isPC s cid uid' ∧ pref s uid' PID ≠ Conflict})

```

```

lemma  $\varphi$ -def2:
 $\varphi$  (Trans s a ou s') =
  (ou = outOK ∧
  (∃ cid uid p uid'. a = Cact (cReview cid uid p PID uid')))
apply(cases a, simp-all) subgoal for x1 by (cases x1, auto) .

```

```

fun  $\chi$  :: act ⇒ bool where
 $\chi$  (Uact (uReview cid uid p pid n rc)) = (pid = PID)
|
 $\chi$  (UUact (uuReview cid uid p pid n rc)) = (pid = PID)
|
 $\chi$  - = False

```

```

lemma  $\chi$ -def2:
 $\chi$  a =
  (∃ cid uid p n rc. a = Uact (uReview cid uid p PID n rc) ∨
  a = UUact (uuReview cid uid p PID n rc))
apply(cases a, simp-all)
subgoal for x2 apply (cases x2, auto) .

```

**subgoal for  $x3$  by (cases  $x3$ , auto) .**

**lemma  $eqExcPID\text{-}step\text{-}\varphi\text{-}imp$ :**

**assumes  $s$ : reach  $s$  and  $ss1$ :  $eqExcPID$   $s$   $s1$**

**and  $PID$ :  $PID \in \in paperIDs$   $s$   $cid$  and  $ph$ : phase  $s$   $cid > revPH$**

**and  $step$ :  $step$   $s$   $a = (ou, s')$  and  $step1$ :  $step$   $s1$   $a = (ou1, s1')$**

**and  $\varphi$ :  $\neg \varphi$  ( $Trans$   $s$   $a$   $ou$   $s'$ )**

**shows  $\neg \varphi$  ( $Trans$   $s1$   $a$   $ou1$   $s1'$ )**

**using  $assms$  unfolding  $\varphi\text{-}def2$  apply (auto simp add: c-defs  $eqExcPID\text{-}imp$ )**

**unfolding  $eqExcPID\text{-}def$**

**apply (metis  $eqExcRLR\text{-}imp[OF$   $s]$   $eqExcRLR\text{-}imp2$ )**

**apply (metis  $eqExcRLR\text{-}imp[OF$   $s]$   $eqExcRLR\text{-}imp2$ )**

**using  $eqExcRLR\text{-}imp[OF$   $s]$   $PID$  by (metis less-not-refl paperIDs-equals)**

**lemma  $eqExcPID\text{-}step\text{-}\varphi$ :**

**assumes reach  $s$  and reach  $s1$  and  $ss1$ :  $eqExcPID$   $s$   $s1$**

**and  $PID$ :  $PID \in \in paperIDs$   $s$   $cid$  and  $ph$ : phase  $s$   $cid > revPH$**

**and  $step$ :  $step$   $s$   $a = (ou, s')$  and  $step1$ :  $step$   $s1$   $a = (ou1, s1')$**

**shows  $\varphi$  ( $Trans$   $s$   $a$   $ou$   $s'$ ) =  $\varphi$  ( $Trans$   $s1$   $a$   $ou1$   $s1'$ )**

**proof –**

**have  $PID \in \in paperIDs$   $s1$   $cid \wedge$  phase  $s1$   $cid > revPH$**

**using  $eqExcPID\text{-}imp[OF$   $ss1]$   $PID$   $ph$  by auto**

**thus ?thesis by (metis  $eqExcPID\text{-}step\text{-}\varphi\text{-}imp$   $eqExcPID\text{-}sym$   $assms$ )**

**qed**

**lemma  $non\text{-}eqExcPID\text{-}step\text{-}\varphi\text{-}imp$ :**

**assumes  $s$ : reach  $s$  and  $ss1$ :  $eqExcPID$   $s$   $s1$**

**and  $PID$ :  $PID \in \in paperIDs$   $s$   $cid$  and  $ou$ :  $ou \neq outErr$**

**and  $step$ :  $step$   $s$   $a = (ou, s')$  and  $step1$ :  $step$   $s1$   $a = (ou1, s1')$**

**and  $\varphi$ :  $\neg \varphi$  ( $Trans$   $s$   $a$   $ou$   $s'$ )**

**shows  $\neg \varphi$  ( $Trans$   $s1$   $a$   $ou1$   $s1'$ )**

**using  $assms$  unfolding  $\varphi\text{-}def2$  by (auto simp add: c-defs  $eqExcPID\text{-}imp$ )**

**lemma  $eqExcPID\text{-}step$ :**

**assumes  $s$ : reach  $s$  and  $s1$ : reach  $s1$**

**and  $ss1$ :  $eqExcPID$   $s$   $s1$**

**and  $step$ :  $step$   $s$   $a = (ou, s')$**

**and  $step1$ :  $step$   $s1$   $a = (ou1, s1')$**

**and  $PID$ :  $PID \in \in paperIDs$   $s$   $cid$**

**and  $ou\text{-}ph$ :  $ou \neq outErr \vee$  phase  $s$   $cid > revPH$**

**and  $\varphi$ :  $\neg \varphi$  ( $Trans$   $s$   $a$   $ou$   $s'$ ) and  $\chi$ :  $\neg \chi$   $a$**

**shows  $eqExcPID$   $s'$   $s1'$**

**proof –**

**have  $s'$ : reach  $s'$  by (metis reach-PairI  $s$   $step$ )**

```

note eqs = eqExcPID-imp[OF ss1]
note eqs' = eqExcPID-imp1[OF ss1]

note eqss = eqExcPID-imp'[OF s ss1]

note_simps[simp] = c-defs u-defs uu-defs r-defs l-defs Paper-dest-conv eqExc-
cPID-def
note_simps2[simp] = eqExcRLR-imp2[where s=s and ?s1.0 = s1] eqExcRLR-imp2[where
s=s' and ?s1.0 = s1]
  eqExcRLR-set[of (roles s cid uid) (roles s1' cid uid) for uid cid]
  eqExcRLR-set[of (roles s' cid uid) (roles s1 cid uid) for uid cid]
  foo2 foo3 eqExcRLR-imp[OF s, where ?s1.0=s1] eqExcRLR-imp[OF s',
where ?s1.0=s1]

note * = step step1 eqs eqs'  $\varphi$   $\chi$  PID ou-ph

then show ?thesis
proof (cases a)
  case (Cact x1)
    with * show ?thesis
    proof (cases x1)
      case (cReview x81 x82 x83 x84 x85)
        with Cact * show ?thesis
        by clarsimp (metis less-irrefl-nat paperIDs-equals s1_simps2(9))
    qed auto
  next
    case (Uact x2)
      with * show ?thesis
      proof (cases x2)
        case (uReview x71 x72 x73 x74 x75 x76)
          with Uact * show ?thesis
          by (clarsimp simp del:_simps2) auto
      qed auto
    next
      case (UUact x3)
        with * show ?thesis
        proof (cases x3)
          case (uuReview x31 x32 x33 x34 x35 x36)
            with UUact * show ?thesis
            by (clarsimp simp del:_simps2) auto
          qed auto
        qed auto
    qed
qed

lemma  $\varphi$ -step-eqExcPID2:
assumes  $\varphi$ :  $\varphi$  (Trans s a ou s')
and s: step s a = (ou, s')
shows eqExcPID2 s s'

```

using  $\varphi$  *cReview-step-eqExcPID2*[*OF* - *s*] unfolding  $\varphi$ -*def2* by *blast*

**lemma** *eqExcPID2-step*:  
**assumes** *s*: *reach s*  
**and** *ss1*: *eqExcPID2 s s1*  
**and** *step*: *step s a = (ou,s<sup>^</sup>)*  
**and** *step1*: *step s1 a = (ou1,s1<sup>^</sup>)*  
**and** *PID*: *PID ∈∈ paperIDs s cid* **and** *ph*: *phase s cid ≥ revPH*  
**and**  $\varphi$ :  $\neg \varphi$  (*Trans s a ou s<sup>^</sup>*)  
**shows** *eqExcPID2 s' s1'*  
**proof** –  
  **have** *s'*: *reach s'* **by** (*metis reach-PairI s step*)  
  **note** *eqs* = *eqExcPID2-imp*[*OF ss1*]  
  
  **note** *eqs'* = *eqExcPID2-imp1*[*OF ss1*]  
  
  **note** *eqss* = *eqExcPID2-imp'*[*OF s ss1*]  
  
  **note** *simps*[*simp*] = *c-defs u-defs uu-defs r-defs l-defs*  
    *Paper-dest-conv*  
    *eqExcPID2-def eqExcPID-def*  
    *eqExcR*  
  **note** *simps2*[*simp*] = *eqExcRLR-imp2*[**where** *s=s* **and** *?s1.0 = s1*  $\wedge$  *eqExcRLR-imp2*[**where**  
*s=s'* **and** *?s1.0 = s1*]  
    *eqExcRLR-set*[*of (roles s cid uid) (roles s1' cid uid) for cid uid*]  
    *eqExcRLR-set*[*of (roles s' cid uid) (roles s1 cid uid) for cid uid*]  
    *foo2 foo3 eqExcRLR-imp*[*OF s, where ?s1.0=s1*  $\wedge$  *eqExcRLR-imp*[*OF s'*,  
**where** *?s1.0=s1*]  
  **note**  $*$  = *step step1 eqs eqs' ph PID*  $\varphi$   
  
  **then show** *?thesis*  
  **proof** (*cases a*)  
    **case** (*Cact x1*)  
    **with**  $*$  **show** *?thesis*  
    **proof** (*cases x1*)  
      **case** (*cReview x81 x82 x83 x84 x85*)  
      **with** *Cact \** **show** *?thesis*  
      **by** *clarsimp (metis simps2(9))*  
    **qed** *auto*  
  **next**  
    **case** (*Uact x2*)  
    **with**  $*$  **show** *?thesis*  
    **proof** (*cases x2*)  
      **case** (*uReview x71 x72 x73 x74 x75 x76*)  
      **with** *Uact \** **show** *?thesis*  
      **by** (*clarsimp simp del: simps2*) *auto*  
    **qed** *auto*  
  **next**  
    **case** (*UUact x3*)

```

with * show ?thesis
proof (cases x3)
  case (wuReview x31 x32 x33 x34 x35 x36)
  with UUact * show ?thesis
  by (clarsimp simp del:_simps2) auto
qed auto
next
case (Lact x5)
with * show ?thesis by (cases x5; auto)
qed auto
qed

```

**lemma** *eqExcPID2-step-φ-imp*:  
**assumes** *s: reach s* **and** *ss1: eqExcPID2 s s1*

**and** *PID: PID ∈∈ paperIDs s cid* **and** *ph: phase s cid > revPH*

**and** *step: step s a = (ou, s')* **and** *step1: step s1 a = (ou1, s1')*  
**and**  $\varphi: \neg \varphi (Trans\ s\ a\ ou\ s')$   
**shows**  $\neg \varphi (Trans\ s1\ a\ ou1\ s1')$   
**using** *assms unfolding φ-def2* **apply** (*auto simp add: c-defs eqExcPID2-imp*)  
**unfolding** *eqExcPID2-def*  
**apply**(*metis eqExcRLR-imp[OF s] eqExcRLR-imp2*)  
**apply**(*metis eqExcRLR-imp[OF s] eqExcRLR-imp2*)  
**using** *eqExcRLR-imp[OF s] PID* **by** (*metis less-not-reft paperIDs-equals*)

**lemma** *eqExcPID2-step-φ*:  
**assumes** *reach s* **and** *reach s1* **and** *ss1: eqExcPID2 s s1*

**and** *PID: PID ∈∈ paperIDs s cid* **and** *ph: phase s cid > revPH*

**and** *step: step s a = (ou, s')* **and** *step1: step s1 a = (ou1, s1')*  
**shows**  $\varphi (Trans\ s\ a\ ou\ s') = \varphi (Trans\ s1\ a\ ou1\ s1')$   
**proof**–  
**have** *PID ∈∈ paperIDs s1 cid ∧ phase s1 cid > revPH*  
**using** *eqExcPID2-imp[OF ss1] PID ph* **by** *auto*  
**thus** *?thesis* **by** (*metis eqExcPID2-step-φ-imp eqExcPID2-sym assms*)  
**qed**

**lemma** *non-eqExcPID2-step-φ-imp*:  
**assumes** *s: reach s* **and** *ss1: eqExcPID2 s s1*  
**and** *PID: PID ∈∈ paperIDs s cid* **and** *ou: ou ≠ outErr*  
**and** *step: step s a = (ou, s')* **and** *step1: step s1 a = (ou1, s1')*  
**and**  $\varphi: \neg \varphi (Trans\ s\ a\ ou\ s')$   
**shows**  $\neg \varphi (Trans\ s1\ a\ ou1\ s1')$   
**using** *assms unfolding φ-def2* **by** (*auto simp add: c-defs eqExcPID2-imp*)

```

end
theory Reviewer-Assignment-NCPC
imports ../Observation-Setup Reviewer-Assignment-Value-Setup Bounded-Deducibility-Security.Compositiona
begin

```

### 9.3 Confidentiality protection from non-PC-members

We verify the following property:

A group of users UIDs learn nothing about the reviewers assigned to a paper PID except for their number and the fact that they are PC members having no conflict with that paper unless/until the user becomes a PC member in the paper's conference having no conflict with that paper and the conference moves to the reviewing phase.

```

fun T :: (state,act,out) trans  $\Rightarrow$  bool where
  T (Trans - - ou s') =
    ( $\exists$  uid  $\in$  UIDs.  $\exists$  cid.
      PID  $\in$  paperIDs s' cid  $\wedge$  isPC s' cid uid  $\wedge$  pref s' uid PID  $\neq$  Conflict  $\wedge$  phase
      s' cid  $\geq$  revPH)

```

```

term isAUT

```

```

declare T.simps [simp del]

```

```

definition B :: value list  $\Rightarrow$  value list  $\Rightarrow$  bool where
  B vl vl1  $\equiv$ 
    vl  $\neq$  []  $\wedge$ 
    length vl = length vl1  $\wedge$ 
    distinct (map fst vl1)  $\wedge$  fst ' (set vl1)  $\subseteq$  snd (hd vl)  $\wedge$  snd ' (set vl1) = {snd (hd
    vl)}

```

```

interpretation BD-Security-IO where
  istate = istate and step = step and
   $\varphi = \varphi$  and  $f = f$  and  $\gamma = \gamma$  and  $g = g$  and  $T = T$  and  $B = B$ 
done

```

```

lemma reachNT-non-isPC-isChair:
assumes reachNT s and uid  $\in$  UIDs
shows
  (PID  $\in$  paperIDs s cid  $\wedge$  isPC s cid uid  $\longrightarrow$ 
    pref s uid PID = Conflict  $\vee$  phase s cid < revPH)  $\wedge$ 

```

$(PID \in \in paperIDs\ s\ cid \wedge isChair\ s\ cid\ uid \longrightarrow$   
 $pref\ s\ uid\ PID = Conflict \vee phase\ s\ cid < revPH)$   
**using** *assms*  
**apply** *induct*  
**subgoal by** (*auto simp: istate-def*)  
**subgoal apply**(*intro conjI*)  
**apply** (*metis T.simps not-le-imp-less trans.collapse*)  
**by** (*metis T.simps isChair-isPC not-le-imp-less reachNT.Step reachNT-reach*  
*trans.collapse*)  
**done**

**lemma** *T-φ-γ*:  
**assumes** *1: reachNT s and 2: step s a = (ou,s') φ (Trans s a ou s')*  
**shows**  $\neg \gamma (Trans\ s\ a\ ou\ s')$   
**using** *reachNT-non-isPC-isChair[OF 1] 2 unfolding T.simps φ-def2*  
**by** (*fastforce simp: c-defs isRev-imp-isRevNth-getReviewIndex*)

**lemma** *T-φ-γ-stronger*:  
**assumes** *s: reach s and 0: PID ∈ ∈ paperIDs s cid*  
**and** *2: step s a = (ou,s') φ (Trans s a ou s')*  
**and** *1: ∀ uid ∈ UIDs. isChair s cid uid ⟶ pref s uid PID = Conflict ∨ phase s*  
*cid < revPH*  
**shows**  $\neg \gamma (Trans\ s\ a\ ou\ s')$   
**proof**–  
**have**  $\neg (\exists uid \in UIDs. \exists cid. PID \in \in paperIDs\ s\ cid \wedge$   
 $isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict \wedge phase\ s\ cid \geq revPH)$   
**using** *0 1 s by (metis not-le paperIDs-equals)*  
**thus** *?thesis using assms unfolding T.simps φ-def2 by (force simp add: c-defs)*  
**qed**

**lemma** *T-φ-γ-1*:  
**assumes** *s: reachNT s and s1: reach s1 and PID: PID ∈ ∈ paperIDs s cid*  
**and** *ss1: eqExcPID s s1*  
**and** *step1: step s1 a = (ou1,s1') and φ1: φ (Trans s1 a ou1 s1')*  
**and**  $\varphi: \neg \varphi (Trans\ s\ a\ ou\ s')$   
**shows**  $\neg \gamma (Trans\ s1\ a\ ou1\ s1')$   
**proof**–  
**have**  $\forall uid \in UIDs. isChair\ s\ cid\ uid \longrightarrow pref\ s\ uid\ PID = Conflict \vee phase\ s$   
 $cid < revPH$   
**using** *reachNT-non-isPC-isChair[OF s] PID by auto*  
**hence** *1: ∀ uid ∈ UIDs. isChair s1 cid uid ⟶ pref s1 uid PID = Conflict ∨*  
*phase s1 cid < revPH*  
**using** *ss1 unfolding eqExcPID-def using eqExcRLR-imp2 by fastforce*  
**have** *PID1: PID ∈ ∈ paperIDs s1 cid using PID ss1 eqExcPID-imp by auto*  
**show** *?thesis apply (rule T-φ-γ-stronger[OF s1 PID1 step1 φ1]) using 1 by*  
*auto*  
**qed**

**lemma** *notIsPCorConflict-eqExcPID-roles-eq*:

**assumes**  $s$ : *reach*  $s$  **and**  $s1$ : *reach*  $s1$  **and**  $PID$ :  $PID \in \in paperIDs\ s\ cid$   
**and**  $pc$ :  $\neg isPC\ s\ cid\ uid \vee pref\ s\ uid\ PID = Conflict$   
**and**  $eeq$ :  $eqExcPID\ s\ s1$   
**shows**  $roles\ s\ cid\ uid = roles\ s1\ cid\ uid$   
**proof** –  
**have**  $eq$ :  $eqExcRLR\ (roles\ s\ cid\ uid)\ (roles\ s1\ cid\ uid)$   
**using**  $eeq$  **unfolding**  $eqExcPID-def$  **by** *auto*  
**have**  $\neg isPC\ s1\ cid\ uid \vee pref\ s1\ uid\ PID = Conflict$   
**using**  $pc$   $eqExcPID-imp[OF\ eeq]$   $eqExcRLR-imp2[OF\ eq]$  **by** *auto*  
**hence**  $\neg isRev\ s\ cid\ uid\ PID \wedge \neg isRev\ s1\ cid\ uid\ PID$  **using**  $pc\ s\ s1\ PID$   
**by** (*metis isRev-pref-notConflict-isPC*)  
**thus**  $?thesis$  **using**  $eq$  **unfolding**  $eqExcRLR-def$   
**by** (*metis Bex-set-list-ex filter-id-conv isRev-def*)  
**qed**

**lemma** *notInPaperIDs-eqExcPID-roles-eq*:  
**assumes**  $s$ : *reach*  $s$  **and**  $s1$ : *reach*  $s1$  **and**  $PID$ :  $\neg PID \in \in paperIDs\ s\ cid$   
**and**  $eq$ :  $eqExcPID\ s\ s1$   
**shows**  $roles\ s\ cid\ uid = roles\ s1\ cid\ uid$   
**proof** –  
**have**  $\neg PID \in \in paperIDs\ s1\ cid$  **using**  $PID$   $eq$  **unfolding**  $eqExcPID-def$  **by** *auto*  
**hence**  $\neg isRev\ s\ cid\ uid\ PID \wedge \neg isRev\ s1\ cid\ uid\ PID$  **using**  $s\ s1\ PID$  **by** (*metis isRev-paperIDs*)  
**thus**  $?thesis$  **using**  $eq$  **unfolding**  $eqExcPID-def$   $eqExcRLR-def$   
**by** (*metis Bex-set-list-ex filter-True isRev-def*)  
**qed**

**lemma** *eqExcPID-step-out*:  
**assumes**  $ss1$ :  $eqExcPID\ s\ s1$   
**and**  $step$ :  $step\ s\ a = (ou, s')$  **and**  $step1$ :  $step\ s1\ a = (ou1, s1')$   
**and**  $sT$ :  $reachNT\ s$  **and**  $s1$ : *reach*  $s1$   
**and**  $PID$ :  $PID \in \in paperIDs\ s\ cid$  **and**  $ph$ :  $phase\ s\ cid \geq revPH$   
**and**  $\varphi$ :  $\neg \varphi\ (Trans\ s\ a\ ou\ s')$  **and**  $\varphi1$ :  $\neg \varphi\ (Trans\ s1\ a\ ou1\ s1')$  **and**  $\chi$ :  $\neg \chi\ a$   
**and**  $UIDs$ :  $userOfA\ a \in UIDs$   
**shows**  $ou = ou1$   
**proof** –  
**note**  $s = reachNT-reach[OF\ sT]$   
**have**  $s'$ : *reach*  $s'$  **and**  $s1'$ : *reach*  $s1'$  **by** (*metis reach-PairI s s1 step step1*)+  
**have**  $s's1'$ :  $eqExcPID\ s'\ s1'$   
**using**  $\chi\ \varphi\ \varphi1$   $eqExcPID-step$   $eqExcPID-sym\ s\ s1\ ss1\ step\ step1\ step-outErr-eq$   
**by** (*metis PID isAut-paperIDs notInPaperIDs-eqExcPID-roles-eq paperID-ex-userID1 paperID-ex-userID-def*)  
**have**  $s1's'$ :  $eqExcPID\ s1'\ s'$  **by** (*metis eqExcPID-sym s's1'*)  
**have**  $s'$ : *reach*  $s'$  **by** (*metis reach-PairI s step*)  
**have**  $ph1$ :  $phase\ s1\ cid \geq revPH$  **using**  $ph\ ss1$  **unfolding**  $eqExcPID-def$  **by** *auto*  
**have**  $ph'$ :  $phase\ s'\ cid \geq revPH$  **and**  $ph1'$ :  $phase\ s1'\ cid \geq revPH$   
**using**  $ph\ ph1$  **by** (*metis dual-order.trans phase-increases step step1*)+  
**note**  $Inv = reachNT-non-isPC-isChair[OF\ sT\ UIDs]$   
**note**  $eqs = eqExcPID-imp[OF\ ss1]$

```

note eqs' = eqExcPID-imp1[OF ss1]

note_simps[simp] = c-defs u-defs uu-defs r-defs l-defs Paper-dest-conv eqExcPID-def
note_simps2[simp] = eqExcRLLR-imp[of s - - s1, OF s] eqExcRLLR-imp[of s' - - s1']
  eqExcRLLR-imp[of s - - s1'] eqExcRLLR-imp[of s' - - s1]
  eqExcRLLR-imp2[of s - - s1] eqExcRLLR-imp2[of s' - - s1']
  eqExcRLLR-imp2[of s - - s1'] eqExcRLLR-imp2[of s' - - s1]
  eqExcRLLR-set[of (roles s cid uid) (roles s1 cid uid) for cid uid]
  eqExcRLLR-set[of (roles s cid uid) (roles s1' cid uid) for cid uid]
  eqExcRLLR-set[of (roles s' cid uid) (roles s1 cid uid) for cid uid]
  eqExcRLLR-set[of (roles s' cid uid) (roles s1' cid uid) for cid uid]
  foo2 foo3
  eqExcRLLR-imp-isRevRole-imp

{fix cid uid p pid assume a = Ract (rMyReview cid uid p pid)
 hence ?thesis
 using step step1 eqs eqs' s s1 UIDs PID  $\varphi$   $\varphi 1$   $\chi$  paperIDs-equals[OF s] Inv
 apply (auto simp add: isRev-getRevRole getRevRole-Some)[]
 apply (metis eqExcPID-imp' isRev-isPC not-less option.inject pref-Conflict-isRev
  role.distinct role.inject s1's'
  isRev-isPC not-less pref-Conflict-isRev_simps2(1)_simps2(8) is-
  Rev-getRevRole getRevRole-Some)+
 done
} note this[simp]

have ?thesis
 using step step1 eqs eqs' s s1 UIDs PID  $\varphi$   $\varphi 1$   $\chi$ 
  paperIDs-equals[OF s] Inv
 apply(cases a, simp-all only:)
  subgoal for x1 apply(cases x1)
    apply auto[] apply auto[] apply auto[] apply auto[]
    apply auto[] apply auto[] apply auto[] apply auto[] .
  subgoal for x2 apply(cases x2)
    apply auto[] apply auto[] apply auto[] apply auto[]
    apply auto[] apply auto[] apply auto[] .
  subgoal for x3 apply(cases x3)
    apply auto[] apply auto[] apply auto[] apply auto[] .
  subgoal for x4 apply(cases x4)
    apply auto[] apply auto[] apply auto[] apply auto[]
    apply auto[] apply auto[] apply auto[] apply auto[]
    apply clarsimp apply (metis eqExcPID-imp2 not-less ph' s's1')
    apply auto[] apply auto[] apply auto[] apply auto[]
    apply auto[] .
  subgoal for x5 apply(cases x5)
    apply auto[] apply auto[] apply auto[]
    apply clarsimp apply (metis (opaque-lifting) empty-iff list.set(1) notInPa-
  perIDs-eqExcPID-roles-eq notIsPCorConflict-eqExcPID-roles-eq s's1' s1's')

```

```

    apply auto[] apply auto[] apply auto[] apply auto[]
    apply auto[] apply auto[]
    apply clarsimp apply (metis Suc-leI filter-cong isRev-pref-notConflict-isPC
not-less-eq-eq_simps2(2))
    apply clarsimp apply (metis not-less_simps2(1)) .
  done

note * = step step1 eqs eqs' s s1 UIDs PID  $\varphi$   $\varphi1$   $\chi$  paperIDs-equals[OF s] Inv

show ?thesis
proof (cases a)
  case (Cact x1)
  with * show ?thesis by (cases x1; auto)
next
  case (Uact x2)
  with * show ?thesis by (cases x2; auto)
next
  case (UUact x3)
  with * show ?thesis by (cases x3; auto)
next
  case (Ract x4)
  with * show ?thesis
proof (cases x4)
  case (rReviews x91 x92 x93 x94)
  with Ract * show ?thesis
  by clarsimp (metis eqExcPID-imp2 not-less ph' s's1')
qed auto
next
  case (Lact x5)
  with * show ?thesis
proof (cases x5)
  case (lMyConfs x41 x42)
  with Lact * show ?thesis
  by clarsimp (metis (opaque-lifting) empty-iff list.set(1) notInPaperIDs-eqExcPID-roles-eq
notIsPCorConflict-eqExcPID-roles-eq s's1' s1's')
next
  case (lMyAssignedPapers x111 x112 x113)
  with Lact * show ?thesis
  by clarsimp (metis Suc-leI filter-cong isRev-pref-notConflict-isPC not-less-eq-eq
_simps2(2))
next
  case (lAssignedReviewers x121 x122 x123 x124)
  with Lact * show ?thesis
  by clarsimp (metis not-less_simps2(1))
qed auto
qed
qed

lemma eqExcPID-step- $\varphi$ -eqExcPID-out:

```

**assumes**  $s$ : reach  $s$  **and**  $s1$ : reach  $s1$   
**and**  $a$ :  $a = \text{Cact } (c\text{Review } cid \text{ uid } p \text{ PID } uid')$   
**and**  $a1$ :  $a1 = \text{Cact } (c\text{Review } cid \text{ uid } p \text{ PID } uid1')$   
**and**  $ss1$ :  $eqExcPID \ s \ s1$  **and**  $step$ :  $step \ s \ a = (outOK, s')$   
**and**  $pc$ :  $isPC \ s \ cid \ uid1' \wedge \text{pref } s \ uid1' \text{ PID} \neq \text{Conflict}$   
**and**  $rv1$ :  $\neg isRev \ s1 \ cid \ uid1' \text{ PID}$  **and**  $step1$ :  $step \ s1 \ a1 = (ou1, s1')$   
**shows**  $eqExcPID \ s' \ s1' \wedge ou1 = outOK$   
**proof** –  
**have**  $s'$ : reach  $s'$  **and**  $s1'$ : reach  $s1'$  **by** (*metis reach-PairI*  $s \ s1 \ step \ step1$ ) +  
**have**  $c$ :  $isChair \ s \ cid \ uid \wedge \text{pref } s \ uid \text{ PID} \neq \text{Conflict} \wedge$   
 $phase \ s \ cid = revPH \wedge \text{pass } s \ uid = p \wedge$   
 $PID \in \in \text{paperIDs } s \ cid \wedge cid \in \in \text{confIDs } s$   
**using**  $step$  **unfolding**  $a$  **by** (*auto simp: c-defs*)  
**hence**  $c1$ :  $isChair \ s1 \ cid \ uid \wedge \text{pref } s1 \ uid \text{ PID} \neq \text{Conflict} \wedge$   
 $phase \ s1 \ cid = revPH \wedge \text{pass } s1 \ uid = p \wedge$   
 $PID \in \in \text{paperIDs } s1 \ cid \wedge cid \in \in \text{confIDs } s1$   
**and**  $pc1$ :  $isPC \ s1 \ cid \ uid1' \wedge \text{pref } s1 \ uid1' \text{ PID} \neq \text{Conflict}$   
**using**  $pc \ ss1$  **unfolding**  $eqExcPID\text{-def}$  **using**  $eqExcRLLR\text{-imp2}$  **by** *metis* +  
  
**note**  $\text{simps}[simp] = c\text{-defs } u\text{-defs } uu\text{-defs } r\text{-defs } l\text{-defs } \text{Paper-dest-conv } eqExc\text{-}cPID\text{-def}$   
**note**  $\text{simps2}[simp] = eqExcRLLR\text{-imp}[of \ s \ \ \ \ s1, OF \ s] \ eqExcRLLR\text{-imp}[of \ s' \ \ \ \ s1' \ \ \ \ s1']$   
 $eqExcRLLR\text{-imp}[of \ s \ \ \ \ s1'] \ eqExcRLLR\text{-imp}[of \ s' \ \ \ \ s1']$   
 $eqExcRLLR\text{-imp2}[of \ s \ \ \ s1] \ eqExcRLLR\text{-imp2}[of \ s' \ \ \ s1']$   
 $eqExcRLLR\text{-imp2}[of \ s \ \ \ s1'] \ eqExcRLLR\text{-imp2}[of \ s' \ \ \ s1']$   
 $eqExcRLLR\text{-set}[of \ (\text{roles } s \ cid \ uid) \ (\text{roles } s1 \ cid \ uid) \ \text{for } cid \ uid]$   
 $eqExcRLLR\text{-set}[of \ (\text{roles } s \ cid \ uid) \ (\text{roles } s1' \ cid \ uid) \ \text{for } cid \ uid]$   
 $eqExcRLLR\text{-set}[of \ (\text{roles } s' \ cid \ uid) \ (\text{roles } s1 \ cid \ uid) \ \text{for } cid \ uid]$   
 $eqExcRLLR\text{-set}[of \ (\text{roles } s' \ cid \ uid) \ (\text{roles } s1' \ cid \ uid) \ \text{for } cid \ uid]$   
 $foo2 \ foo3$   
 $eqExcRLLR\text{-imp-isRevRole-imp}$   
  
**show** *?thesis*  
**using**  $pc1 \ c1 \ ss1 \ step \ step1 \ c \ c1 \ pc \ rv1$  **unfolding**  $eqExcPID\text{-def}$   $a \ a1$   
**using**  $roles\text{-userIDs } s1'$  **by** *force*  
**qed**

**lemma**  $eqExcPID\text{-ex-isNthReview}$ :  
**assumes**  $s$ : reach  $s$  **and**  $s1$ : reach  $s1$  **and**  $e$ :  $eqExcPID \ s \ s1$   
**and**  $i$ :  $isRevNth \ s \ cid \ uid \text{ PID } n$   
**shows**  $\exists uid1. isRevNth \ s1 \ cid \ uid1 \text{ PID } n$   
**proof** –  
**have**  $PID$ :  $PID \in \in \text{paperIDs } s \ cid$  **by** (*metis i isRevNth-paperIDs s*)  
**have**  $n < \text{length } (\text{reviewsPaper } (\text{paper } s \text{ PID}))$  **using**  $s \ i$  **by** (*metis isRevNth-less-length*)  
**hence**  $PID \in \in \text{paperIDs } s1 \ cid \wedge n < \text{length } (\text{reviewsPaper } (\text{paper } s1 \text{ PID}))$   
**using**  $e \ PID$  **unfolding**  $eqExcPID\text{-def}$  **by** *auto*  
**thus** *?thesis* **using**  $s1$  **by** (*metis reviews-compact*)  
**qed**

**lemma** *eqExcPID-step- $\chi$ 1*:  
**assumes**  $s$ : *reach s* **and**  $s1$ : *reach s1*  
**and**  $a$ :  $a = Uact (uReview\ cid\ uid\ p\ PID\ n\ rc)$   
**and**  $ss1$ : *eqExcPID s s1* **and**  $step$ :  $step\ s\ a = (outOK, s')$   
**shows**  
 $\exists\ s1'\ uid1\ p.$   
 $isRevNth\ s1\ cid\ uid1\ PID\ n\ \wedge$   
 $step\ s1\ (Uact\ (uReview\ cid\ uid1\ p\ PID\ n\ rc)) = (outOK, s1')\ \wedge$   
 $eqExcPID\ s'\ s1'$

**proof** –  
**have**  $s'$ : *reach s'* **by** (*metis reach-PairI s step*)  
**have**  $c$ :  $isRevNth\ s\ cid\ uid\ PID\ n\ \wedge\ phase\ s\ cid = revPH\ \wedge\ PID\ \in\in\ paperIDs\ s$   
 $cid\ \wedge\ cid\ \in\in\ confIDs\ s$   
**using**  $step$  **unfolding**  $a$  **apply** (*auto simp: u-defs*) **by** (*metis isRev-imp-isRevNth-getReviewIndex*)  
**obtain**  $uid1$  **where**  $rv1$ :  $isRevNth\ s1\ cid\ uid1\ PID\ n$  **using**  $s\ s1\ ss1$  **by** (*metis*  
 $c\ eqExcPID-ex-isNthReview$ )  
**let**  $?p = pass\ s1\ uid1$   
**define**  $a1$  **where**  $a1$ :  $a1 \equiv Uact (uReview\ cid\ uid1\ (pass\ s1\ uid1)\ PID\ n\ rc)$   
**obtain**  $ou1\ s1'$  **where**  $step1$ :  $step\ s1\ a1 = (ou1, s1')$  **by** (*metis surj-pair*)  
**have**  $s1'$ : *reach s1'* **by** (*metis reach-PairI s1 step1*)  
**have**  $c1$ :  $phase\ s1\ cid = revPH\ \wedge\ PID\ \in\in\ paperIDs\ s1\ cid\ \wedge\ cid\ \in\in\ confIDs\ s1$   
**using**  $ss1\ c$  **unfolding** *eqExcPID-def* **using** *eqExcRLR-imp2* **by** *auto*  
**show**  $?thesis$   
**apply** (*intro exI[of - s1'] exI[of - uid1] exI[of - ?p]*)  
**using**  $rv1\ c1\ ss1\ step\ step1\ c\ c1\ rv1$  **unfolding** *eqExcPID-def a a1*  
**using** *isRevNth-getReviewIndex isRev-def2 roles-userIDs s1'*  
**by** ((*auto*  
 $simp: u-defs$   
 $simp: Paper-dest-conv$   
 $simp: eqExcPID-def$   
))  
**qed**

**lemma** *eqExcPID-step- $\chi$ 2*:  
**assumes**  $s$ : *reach s* **and**  $s1$ : *reach s1*  
**and**  $a$ :  $a = UUact (uuReview\ cid\ uid\ p\ PID\ n\ rc)$   
**and**  $ss1$ : *eqExcPID s s1* **and**  $step$ :  $step\ s\ a = (outOK, s')$   
**shows**  
 $\exists\ s1'\ uid1\ p.$   
 $isRevNth\ s1\ cid\ uid1\ PID\ n\ \wedge$   
 $step\ s1\ (UUact\ (uuReview\ cid\ uid1\ p\ PID\ n\ rc)) = (outOK, s1')\ \wedge$   
 $eqExcPID\ s'\ s1'$

**proof** –  
**have**  $s'$ : *reach s'* **by** (*metis reach-PairI s step*)  
**have**  $c$ :  $isRevNth\ s\ cid\ uid\ PID\ n\ \wedge\ phase\ s\ cid = disPH\ \wedge\ PID\ \in\in\ paperIDs\ s$   
 $cid\ \wedge\ cid\ \in\in\ confIDs\ s$   
**using**  $step$  **unfolding**  $a$  **apply** (*auto simp: uu-defs*) **by** (*metis isRev-imp-isRevNth-getReviewIndex*)  
**obtain**  $uid1$  **where**  $rv1$ :  $isRevNth\ s1\ cid\ uid1\ PID\ n$  **using**  $s\ s1\ ss1$  **by** (*metis*

*c eqExcPID-ex-isNthReview*)  
**let**  $?p = \text{pass } s1 \text{ uid1}$   
**define**  $a1$  **where**  $a1: a1 \equiv UUact (uuReview \text{ cid } uid1 (\text{pass } s1 \text{ uid1}) PID \text{ n } rc)$   
**obtain**  $ou1 \ s1'$  **where**  $step1: step \ s1 \ a1 = (ou1, s1')$  **by**  $(metis \ surj\text{-}pair)$   
**have**  $s1'$ :  $reach \ s1'$  **by**  $(metis \ reach\text{-}PairI \ s1 \ step1)$   
**have**  $c1: phase \ s1 \ cid = disPH \wedge PID \in \in paperIDs \ s1 \ cid \wedge cid \in \in confIDs \ s1$   
**using**  $ss1 \ c$  **unfolding**  $eqExcPID\text{-}def$  **using**  $eqExcRLR\text{-}imp2$  **by**  $auto$   
**show**  $?thesis$   
**apply**  $(intro \ exI[of \ - \ s1'] \ exI[of \ - \ uid1] \ exI[of \ - \ ?p])$   
**using**  $rv1 \ c1 \ ss1 \ step \ step1 \ c \ c1 \ rv1$  **unfolding**  $eqExcPID\text{-}def \ a \ a1$   
**using**  $isRevNth\text{-}getReviewIndex \ isRev\text{-}def2 \ roles\text{-}userIDs \ s1'$   
**by**  $((force$   
 $\quad simp: \ uu\text{-}defs$   
 $\quad simp: \ Paper\text{-}dest\text{-}conv$   
 $\quad simp: \ eqExcPID\text{-}def$   
 $\quad ))$   
**qed**

**definition**  $\Delta1 :: state \Rightarrow value \ list \Rightarrow state \Rightarrow value \ list \Rightarrow bool$  **where**  
 $\Delta1 \ s \ vl \ s1 \ vl1 \equiv$   
 $(\forall \ cid. PID \in \in paperIDs \ s \ cid \longrightarrow phase \ s \ cid < revPH) \wedge s = s1$   
 $\wedge B \ vl \ vl1$

**definition**  $\Delta2 :: state \Rightarrow value \ list \Rightarrow state \Rightarrow value \ list \Rightarrow bool$  **where**  
 $\Delta2 \ s \ vl \ s1 \ vl1 \equiv$   
 $\exists \ cid \ uid.$   
 $PID \in \in paperIDs \ s \ cid \wedge phase \ s \ cid = revPH \wedge$   
 $isChair \ s \ cid \ uid \wedge pref \ s \ uid \ PID \neq Conflict \wedge$   
 $eqExcPID \ s \ s1 \wedge$   
 $length \ vl = length \ vl1 \wedge$   
 $distinct \ (map \ fst \ vl1) \wedge$   
 $fst \ ' (set \ vl1) \subseteq \{uid'. isPC \ s \ cid \ uid' \wedge pref \ s \ uid' \ PID \neq Conflict\} \wedge$   
 $fst \ ' (set \ vl1) \cap \{uid'. isRev \ s1 \ cid \ uid' \ PID\} = \{\}$   $\wedge$   
 $snd \ ' (set \ vl1) \subseteq \{\{uid'. isPC \ s \ cid \ uid' \wedge pref \ s \ uid' \ PID \neq Conflict\}\}$

**definition**  $\Delta3 :: state \Rightarrow value \ list \Rightarrow state \Rightarrow value \ list \Rightarrow bool$  **where**  
 $\Delta3 \ s \ vl \ s1 \ vl1 \equiv$   
 $\exists \ cid. PID \in \in paperIDs \ s \ cid \wedge phase \ s \ cid > revPH \wedge eqExcPID \ s \ s1 \wedge vl1 =$   
 $\square$

**definition**  $\Delta e :: state \Rightarrow value \ list \Rightarrow state \Rightarrow value \ list \Rightarrow bool$  **where**  
 $\Delta e \ s \ vl \ s1 \ vl1 \equiv$   
 $vl \neq \square \wedge$   
 $($   
 $\quad (\exists \ cid. PID \in \in paperIDs \ s \ cid \wedge phase \ s \ cid \geq revPH \wedge$   
 $\quad \quad \neg (\exists \ uid. isChair \ s \ cid \ uid \wedge pref \ s \ uid \ PID \neq Conflict))$   
 $\vee$   
 $\quad (\exists \ cid. PID \in \in paperIDs \ s \ cid \wedge phase \ s \ cid \geq revPH \wedge$

```

      snd (hd vl) ≠ {uid'. isPC s cid uid' ∧ pref s uid' PID ≠ Conflict})
    ∨
    (∃ cid. PID ∈∈ paperIDs s cid ∧ phase s cid > revPH)
  )

lemma istate-Δ1:
  assumes B: B vl vl1
  shows Δ1 istate vl istate vl1
  using B unfolding Δ1-def B-def istate-def by auto

lemma unwind-cont-Δ1: unwind-cont Δ1 {Δ1,Δ2,Δe}
  proof(rule, simp)
    let ?Δ = λs vl s1 vl1. Δ1 s vl s1 vl1 ∨ Δ2 s vl s1 vl1 ∨ Δe s vl s1 vl1
    fix s s1 :: state and vl vl1 :: value list
    assume rsT: reachNT s and rs1: reach s1 and Δ1 s vl s1 vl1
    hence rs: reach s and ss1: s1 = s and B: B vl vl1
    and l: length vl = length vl1
    and c-d: distinct (map fst vl1) ∧ fst ' (set vl1) ⊆ snd (hd vl) ∧ snd ' (set vl1)
    = {snd (hd vl)}
    and vl: vl ≠ []
    and PID-ph: ∧ cid. PID ∈∈ paperIDs s cid ⇒ phase s cid < revPH
    using reachNT-reach unfolding Δ1-def B-def by auto
    have rv: ∧ cid. ¬ (∃ uid'. isRev s cid uid' PID) using rs PID-ph
    by (metis isRev-geq-revPH isRev-paperIDs less-Suc-eq-le not-less-eq-eq)
    show iaction ?Δ s vl s1 vl1 ∨
      ((vl = [] → vl1 = []) ∧ reaction ?Δ s vl s1 vl1) (is ?iact ∨ (- ∧ ?react))
  proof-
    have ?react proof
      fix a :: act and ou :: out and s' :: state and vl'
      let ?trn = Trans s a ou s' let ?trn1 = Trans s1 a ou s'
      assume step: step s a = (ou, s') and T: ¬ T ?trn and c: consume ?trn vl vl'
      have φ: ¬ φ ?trn
      apply(cases a)
      subgoal for x1 apply(cases x1) using step PID-ph by (fastforce simp:
c-defs)+
      by simp-all
      hence vl': vl' = vl using c unfolding consume-def by auto
      show match ?Δ s s1 vl1 a ou s' vl' ∨ ignore ?Δ s s1 vl1 a ou s' vl' (is ?match
∨ ?ignore)
    proof-
      have ?match proof
        show validTrans ?trn1 unfolding ss1 using step by simp
      next
        show consume ?trn1 vl1 vl1 unfolding consume-def ss1 using φ by auto
      next
        show γ ?trn = γ ?trn1 unfolding ss1 by simp
      next
        assume γ ?trn thus g ?trn = g ?trn1 unfolding ss1 by simp
      next
  
```

```

show ? $\Delta$   $s' vl' s' vl1$ 
proof(cases  $\exists cid. PID \in \in paperIDs s cid$ )
  case False note  $PID = False$ 
  have  $PID-ph': \bigwedge cid. PID \in \in paperIDs s' cid \implies phase s' cid < revPH$ 
using  $PID step rs$ 
  apply(cases a)
    subgoal for -  $x1$  apply(cases  $x1$ ) apply(fastforce simp: c-defs) $+$  .
    subgoal for -  $x2$  apply(cases  $x2$ ) apply(fastforce simp: u-defs) $+$  .
    subgoal for -  $x3$  apply(cases  $x3$ ) apply(fastforce simp: uu-defs) $+$  .
    by auto
  hence  $\Delta 1 s' vl' s' vl1$  unfolding  $\Delta 1-def B-def vl'$  using  $PID-ph' c-d vl$ 
l by auto
  thus ?thesis by auto
next
  case True
  then obtain  $CID$  where  $PID: PID \in \in paperIDs s CID$  by auto
  hence  $ph: phase s CID < revPH$  using  $PID-ph$  by auto
  have  $PID': PID \in \in paperIDs s' CID$  by (metis PID paperIDs-mono
step)
  show ?thesis
  proof(cases  $phase s' CID < revPH$ )
    case True note  $ph' = True$ 
    hence  $\Delta 1 s' vl' s' vl1$  unfolding  $\Delta 1-def B-def vl'$  using  $vl c-d ph'$ 
PID' l apply auto
    by (metis reach-PairI paperIDs-equals rs step)
    thus ?thesis by auto
  next
  case False
  hence  $ph-rv': phase s' CID = revPH \wedge \neg (\exists uid'. isRev s' CID uid')$ 
PID)
  using  $ph PID step rs rv$ 
  apply(cases a)
    subgoal for  $x1$  apply(cases  $x1$ ) apply(fastforce simp: c-defs) $+$  .
    subgoal for  $x2$  apply(cases  $x2$ ) apply(fastforce simp: u-defs
isRev-def2) $+$  .
    subgoal for  $x3$  apply(cases  $x3$ ) apply(fastforce simp: uu-defs) $+$  .
    by auto
  show ?thesis
  proof(cases ( $\exists uid. isChair s' CID uid \wedge pref s' uid PID \neq Conflict$ )
 $\wedge$ 
   $snd (hd vl) = \{uid'. isPC s' CID uid' \wedge pref s' uid' PID \neq$ 
Conflict})
    case True
    hence  $\Delta 2 s' vl' s' vl1$ 
    unfolding  $\Delta 2-def B-def vl'$  using  $c-d ph-rv' PID' l$  by auto
    thus ?thesis by auto
  next
  case False
  hence  $\Delta e s' vl' s' vl1$ 

```

```

      unfolding  $\Delta e\text{-def}$   $vl'$  using  $vl$   $c\text{-d}$   $ph\text{-rv}'$   $PID'$   $l$  by auto
      thus ?thesis by auto
    qed
  qed
  qed
  qed
  thus ?thesis by simp
  qed
  qed
  thus ?thesis using  $vl$  by auto
  qed
  qed

```

**lemma** *not- $\varphi$ -isRev-isPC-persists*:

**assumes** *reach*  $s$

$PID \in \in paperIDs$   $s$  *cid* **and**  $\neg \varphi$  (*Trans*  $s$   $a$  *ou*  $s'$ )

**and** *step*  $s$   $a = (ou, s')$

**shows** *isRev*  $s'$  *cid* *uid*  $PID = isRev$   $s$  *cid* *uid*  $PID \wedge isPC$   $s'$  *cid* *uid*  $= isPC$   $s$  *cid* *uid*

**using** *assms* **apply**(*cases*  $a$ )

**subgoal for**  $x1$  **apply**(*cases*  $x1$ ) **apply**(*auto simp: c-defs isRev-def2 roles-confIDs paperIDs-confIDs*)

**apply** (*metis Suc-n-not-le-n paperIDs-geq-subPH*) $+$  .

**subgoal for**  $x2$  **apply**(*cases*  $x2$ , *auto simp: u-defs isRev-def2*) .

**subgoal for**  $x3$  **apply**(*cases*  $x3$ , *auto simp: uu-defs isRev-def2*) .

**by** *auto*

**lemma**  *$\gamma$ -not $\chi$ -eqButPID-outErr*:

**assumes**  $sT$ : *reachNT*  $s$  **and**  $s1$ : *reach*  $s1$

**and**  $UIDs$ : *userOfA*  $a \in UIDs$  **and** *step*: *step*  $s$   $a = (outErr, s')$

**and**  $ss1$ : *eqExcPID*  $s$   $s1$  **and**  $PID$ :  $PID \in \in paperIDs$   $s$   $CID$

**shows** *step*  $s1$   $a = (outErr, s1)$

**proof**–

**have**  $s$ : *reach*  $s$  **by** (*metis reachNT-reach*  $sT$ )

**have** *step*: *step*  $s$   $a = (outErr, s)$  **using** *step* **by** (*metis step-outErr-eq*)

**note**  $Inv = reachNT\text{-non-isPC-isChair}$ [*OF*  $sT$   $UIDs$ ]

**note**  $eqs = eqExcPID\text{-imp}$ [*OF*  $ss1$ ]

**note**  $eqs' = eqExcPID\text{-imp1}$ [*OF*  $ss1$ ]

**have**  $PID1$ :  $PID \in \in paperIDs$   $s1$   $CID$  **using**  $ss1$   $PID$  **unfolding** *eqExcPID-def* **by** *auto*

**note**  $simps[simp] = c\text{-defs } u\text{-defs } uu\text{-defs } r\text{-defs } l\text{-defs } Paper\text{-dest-conv } eqExcPID\text{-def}$

**note**  $simps2[simp] = eqExcRLR\text{-imp}$ [*of*  $s$  - -  $s1$ , *OF*  $s$ ]

*eqExcRLR-imp2*[*of*  $s$  - -  $s1$ ]

*eqExcRLR-set*[*of* (*roles*  $s$  *cid* *uid*) (*roles*  $s1$  *cid* *uid*) **for** *cid* *uid*]

*foo2* *foo3*

*eqExcRLR-imp-isRevRole-imp*

```

note * = step eqs eqs' s s1 UIDs PID PID1 paperIDs-equals[OF s] Inv

show ?thesis
proof (cases a)
  case (Cact x1)
    with * show ?thesis
    by (cases x1; auto; metis less-irrefl-nat_simps2(1))
  next
    case (Uact x2)
    with * show ?thesis
    by (cases x2; auto; metis isRev-pref-notConflict-isPC less-irrefl-nat_simps2(1))
  next
    case (UUact x3)
    with * show ?thesis
    by (cases x3; auto; metis eqs isRev-isPC less-Suc-eq less-irrefl-nat pref-Conflict-isRev
simps2(1))
  next
    case (Ract x4)
    with * show ?thesis
    by (cases x4; auto; metis isRev-isPC not-less pref-Conflict-isRev s1_simps2(1))
  next
    case (Lact x5)
    with * show ?thesis
    by (cases x5; auto)
qed
qed

```

```

lemma exists-failedAct:
 $\exists a. \text{step } s \ a = (\text{outErr}, s) \wedge \text{userOfA } a = \text{uid}$ 
proof -
  obtain p where p: pass s uid = Password p
  by (cases pass s uid)
  let ?a = Cact (cConf undefined uid (Password (fresh {p} p)) undefined undefined)
  show ?thesis apply(rule exI[of - ?a]) using p fresh-notIn[of {p} p] by(auto
simp: c-defs)
qed

```

```

lemma unwind-cont- $\Delta 2$ : unwind-cont  $\Delta 2$  { $\Delta 2, \Delta 3, \Delta e$ }
proof(rule, simp)
  let ? $\Delta$  =  $\lambda s \ vl \ s1 \ vl1. \Delta 2 \ s \ vl \ s1 \ vl1 \vee \Delta 3 \ s \ vl \ s1 \ vl1 \vee \Delta e \ s \ vl \ s1 \ vl1$ 
  fix s s1 :: state and vl vl1 :: value list
  assume rsT: reachNT s and rs1: reach s1 and  $\Delta 2 \ s \ vl \ s1 \ vl1$ 
  then obtain CID uidc where uidc: isChair s CID uidc  $\wedge$  pref s uidc PID  $\neq$  Conflict
  and rs: reach s and ph: phase s CID = revPH (is ?ph = -) and ss1: eqExcPID s s1
  and PID: PID  $\in \in$  paperIDs s CID
  and dis: distinct (map fst vl1)
  and l: length vl = length vl1

```

```

and fst-isPC: fst ‘ (set vl1)  $\subseteq$  {uid'. isPC s CID uid'  $\wedge$  pref s uid' PID  $\neq$  Conflict}
and fst-isRev: fst ‘ (set vl1)  $\cap$  {uid'. isRev s1 CID uid' PID} = {}
and snd-isPC: snd ‘ (set vl1)  $\subseteq$  {{uid'. isPC s CID uid'  $\wedge$  pref s uid' PID  $\neq$  Conflict}}
using reachNT-reach unfolding  $\Delta$ 2-def by auto
hence uidc-notin: uidc  $\notin$  UIDs using less-not-refl3 reachNT-non-isPC-isChair
rsT by fastforce
note vl1-all = dis fst-isPC fst-isRev snd-isPC
have PID1: PID  $\in$  paperIDs s1 CID using PID ss1 unfolding eqExcPID-def
by auto
show iaction ? $\Delta$  s vl s1 vl1  $\vee$ 
  ((vl = []  $\longrightarrow$  vl1 = []))  $\wedge$  reaction ? $\Delta$  s vl s1 vl1 (is ?iact  $\vee$  (-  $\wedge$  ?react))
proof–
  have ?react proof
    fix a :: act and ou :: out and s' :: state and vl'
    let ?trn = Trans s a ou s'
    let ?ph' = phase s' CID
    assume step: step s a = (ou, s') and T:  $\neg T ?trn$  and c: consume ?trn vl vl'
    have PID': PID  $\in$  paperIDs s' CID using PID rs by (metis paperIDs-mono
step)
    have uidc': isChair s' CID uidc  $\wedge$  pref s' uidc PID  $\neq$  Conflict
    using uidc step rs ph PID isChair-persistent revPH-pref-persists[OF rs PID]
by auto
    show match ? $\Delta$  s s1 vl1 a ou s' vl'  $\vee$  ignore ? $\Delta$  s s1 vl1 a ou s' vl' (is ?match
 $\vee$  ?ignore)
    proof(cases  $\varphi$  ?trn)
      case False note  $\varphi = False$ 
      have vl': vl' = vl using c  $\varphi$  unfolding consume-def by (cases vl) auto
      obtain ou1 and s1' where step1: step s1 a = (ou1, s1') by (metis
prod.exhaust)
      let ?trn1 = Trans s1 a ou1 s1'
      show ?thesis
      proof(cases ou = outErr)
        case True note ou = True
        have s': s' = s by (metis ou step step-outErr-eq)
        show ?thesis
        proof (cases userOfA a  $\in$  UIDs)
          case True note uidUIDs = True
          hence ou1: ou1 = outErr using  $\gamma$ -not $\chi$ -eqButPID-outErr
          by (metis PID Pair-inject ou rs1 rsT ss1 step step1)
          hence s1': s1' = s1 by (metis step1 step-outErr-eq)
          have  $\varphi$ 1:  $\neg \varphi ?trn1$  unfolding  $\varphi$ -def2 ou1 by auto
          have ?match proof
            show validTrans ?trn1 using step1 by simp
          next
          show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi$ 1 by auto
          next
          show  $\gamma ?trn = \gamma ?trn1$  by simp

```

```

next
  assume  $\gamma$  ?trn thus  $g$  ?trn =  $g$  ?trn1 unfolding ou ou1 by simp
next
  show ? $\Delta$   $s'$   $vl'$   $s1'$   $vl1$  unfolding  $s'$   $s1'$   $vl'$  by (metis  $\langle \Delta 2$   $s$   $vl$   $s1$   $vl1 \rangle$ )
qed
thus ?thesis by simp
next
  case False note uidUIDs = False
  obtain a1 where ua1: userOfA a1 = userOfA a and step1: step s1 a1
= (outErr, s1)
  by (metis exists-failedAct ou)
  let ?trn1 = Trans s1 a1 outErr s1
  have ?match proof
    show validTrans ?trn1 using step1 by simp
  next
  show consume ?trn1 vl1 vl1 unfolding consume-def  $\varphi$ -def2 ou by auto
  next
  show  $\gamma$  ?trn =  $\gamma$  ?trn1 by (simp add: ua1)
  next
  assume  $\gamma$  ?trn thus  $g$  ?trn =  $g$  ?trn1 using uidUIDs unfolding ou
by simp
  next
  show ? $\Delta$   $s'$   $vl'$   $s1$   $vl1$  unfolding  $s'$   $vl'$  by (metis  $\langle \Delta 2$   $s$   $vl$   $s1$   $vl1 \rangle$ )
  qed
  thus ?thesis by simp
qed
next
  case False note ou = False
  show ?thesis
  proof(cases  $\chi$  a)
  case False note  $\chi$  = False
  have  $s's1'$ : eqExcPID  $s'$   $s1'$  using eqExcPID-step rs rs1 ss1 step step1
PID  $\varphi$   $\chi$  ou by blast
  have  $\varphi 1$ :  $\neg \varphi$  ?trn1 using  $\varphi$  using non-eqExcPID-step- $\varphi$ -imp rs rs1 ss1
PID step step1 ou by auto
  have out: userOfA a  $\in$  UIDs  $\longrightarrow$  ou1 = ou
  using eqExcPID-step-out[OF ss1 step step1 rsT rs1 PID -  $\varphi$   $\varphi 1$   $\chi$ ] ph
by auto
  have ?match proof
    show validTrans ?trn1 using step1 by simp
  next
  show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi 1$  by auto
  next
  show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
  next
  assume  $\gamma$  ?trn thus  $g$  ?trn =  $g$  ?trn1 using out by simp
  next
  show ? $\Delta$   $s'$   $vl'$   $s1'$   $vl1$ 
  proof(cases phase  $s'$  CID = revPH)

```

```

      case True
      hence  $\Delta 2$   $s' vl' s1' vl1$  using  $uidc' PID' s's1' vl1$ -all  $l$  unfolding
 $\Delta 2$ -def  $vl'$ 
      apply(intro  $exI$ [of -  $CID$ ]  $exI$ [of -  $uidc$ ]) apply simp apply(intro
conjI)
      using  $isPC$ -persistent[ $OF$  -  $step$ ]  $revPH$ -pref-persists[ $OF$   $rs$   $PID$  -
step]  $ph$ 
      not- $\varphi$ -isRev-isPC-persists[ $OF$   $rs1$   $PID1$   $\varphi1$   $step1$ ]
       $revPH$ -pref-persists[ $OF$   $rs$   $PID$  -  $step$ ] not- $\varphi$ -isRev-isPC-persists[ $OF$ 
 $rs$   $PID$   $\varphi$   $step$ ]  $ph$ 
      by auto
      thus ? $\Delta$   $s' vl' s1' vl1$  by simp
    next
    case False
      hence  $ph'$ :  $phase$   $s' CID > revPH$  using  $rs$   $step$   $ph$  by ( $metis$ 
antisym-conv2  $phase$ -increases)
      show ?thesis
      proof(cases  $vl = []$ )
      case True hence  $vl1 = []$  using  $l$  by simp
      hence  $\Delta 3$   $s' vl' s1' vl1$  using  $uidc' PID' s's1' ph'$  unfolding
 $\Delta 3$ -def  $vl'$  by auto
      thus ?thesis by simp
    next
    case False
      hence  $\Delta e$   $s' vl' s1' vl1$  using  $PID' ph'$  unfolding  $\Delta e$ -def  $vl'$  by
auto
      thus ?thesis by simp
    qed
  qed
  thus ?thesis by simp
next
case True
thus ?thesis unfolding  $\chi$ -def2 proof( $elim$   $exE$   $disjE$ )
  fix  $cid$   $uid$   $p$   $n$   $rc$  assume  $a$ :  $a = Uact$  ( $uReview$   $cid$   $uid$   $p$   $PID$   $n$   $rc$ )
  hence  $ou$ :  $ou = outOK$  using  $step$   $ou$  unfolding  $a$  by ( $auto$   $simp$ :
 $u$ -defs)
  obtain  $s1' uid1 p$  where
   $uid1$ :  $isRevNth$   $s1$   $cid$   $uid1$   $PID$   $n$ 
  and  $step1$ :  $step$   $s1$  ( $Uact$  ( $uReview$   $cid$   $uid1$   $p$   $PID$   $n$   $rc$ )) = ( $outOK, s1'$ )
(is  $step$  - ? $a1 = -$ )
  and  $s's1'$ :  $eqExcPID$   $s' s1'$  using  $eqExcPID$ -step- $\chi1$   $rs$   $rs1$   $a$   $ss1$   $step$ 
 $ou$  by  $metis$ 
  let ? $trn1 = Trans$   $s1$  ? $a1$   $outOK$   $s1'$ 
  have  $\varphi1$ :  $\neg \varphi$  ? $trn1$  by simp
  have  $isPC$   $s$   $cid$   $uid \wedge pref$   $s$   $uid$   $PID \neq Conflict$ 
  using  $step$  unfolding  $a$   $ou$  apply( $auto$   $simp$ :  $u$ -defs)
  by ( $metis$   $isRev$ -pref-notConflict-isPC  $rs$ )
  hence  $uidUIDs$ :  $\neg uid \in UIDs$  using  $ph$   $reachNT$ -non-isPC-isChair[ $OF$ 

```

```

rsT] apply auto
      by (metis PID PID1 isRevNth-paperIDs less-irrefl-nat paperIDs-equals
rs1 uid1)
      have isPC s1 cid uid1 ∧ pref s1 uid1 PID ≠ Conflict using step1
apply(auto simp: u-defs)
      by (metis isRev-pref-notConflict-isPC rs1)+
      hence isPC s cid uid1 ∧ pref s uid1 PID ≠ Conflict using ss1
unfolding eqExcPID-def
      using eqExcRLR-imp2 by fastforce
      hence uid1UIDs: ¬ uid1 ∈ UIDs using ph reachNT-non-isPC-isChair[OF
rsT] apply auto
      by (metis (no-types, opaque-lifting) eqExcPID-def isRevNth-geq-revPH
isRevNth-paperIDs not-le rs1 ss1 uid1)
      have ph': phase s' CID = revPH using ph step unfolding a by (auto
simp: u-defs)
      have ?match proof
      show validTrans ?trn1 using step1 by simp
next
      show consume ?trn1 vl1 vl1 unfolding consume-def using φ1 by
auto
next
      show γ ?trn = γ ?trn1 using uidUIDs uid1UIDs unfolding a by
simp
next
      assume γ ?trn thus g ?trn = g ?trn1 using uidUIDs unfolding a
by simp
next
      have Δ2 s' vl' s1' vl1 using ph' PID' s's1' unfolding Δ2-def vl'
      apply(intro exI[of - CID] exI[of - uidc]) using l vl1-all
      using isPC-persistent[OF - step] revPH-pref-persists[OF rs PID -
step] ph
      not-φ-isRev-isPC-persists[OF rs1 PID1 φ1 step1]
      revPH-pref-persists[OF rs PID - step] not-φ-isRev-isPC-persists[OF
rs PID φ step] ph uidc'
      by auto
      thus ?Δ s' vl' s1' vl1 by simp
qed
      thus ?thesis by simp
next
      fix cid uid p n rc assume a: a = UUact (uuReview cid uid p PID n rc)
      hence ou: ou = outOK using step ou unfolding a by (auto simp:
uu-defs)
      obtain s1' uid1 p where
      uid1: isRevNth s1 cid uid1 PID n
      and step1: step s1 (UUact (uuReview cid uid1 p PID n rc)) =
(outOK,s1') (is step - ?a1 = -)
      and s's1': eqExcPID s' s1' using eqExcPID-step-χ2 rs rs1 a ss1 step
ou by metis
      let ?trn1 = Trans s1 ?a1 outOK s1'

```

```

      have  $\varphi 1$ :  $\neg \varphi$  ?trn1 by simp
      have isPC s cid uid  $\wedge$  pref s uid PID  $\neq$  Conflict
      using step unfolding a ou apply(auto simp: uu-defs)
      by (metis isRev-pref-notConflict-isPC rs)+
      hence uidUIDs:  $\neg$  uid  $\in$  UIDs using ph reachNT-non-isPC-isChair[OF
rsT] apply auto
      by (metis (no-types, opaque-lifting) eqExcPID-def isRevNth-geq-revPH
isRevNth-paperIDs not-le rs1 ss1 uid1)
      have isPC s1 cid uid1  $\wedge$  pref s1 uid1 PID  $\neq$  Conflict using step1
apply(auto simp: uu-defs)
      by (metis isRev-pref-notConflict-isPC rs1)+
      hence isPC s cid uid1  $\wedge$  pref s uid1 PID  $\neq$  Conflict using ss1
unfolding eqExcPID-def
      using eqExcRLR-imp2 by fastforce
      hence uid1UIDs:  $\neg$  uid1  $\in$  UIDs using ph reachNT-non-isPC-isChair[OF
rsT] apply auto
      by (metis (no-types, opaque-lifting) eqExcPID-def isRevNth-geq-revPH
isRevNth-paperIDs not-le rs1 ss1 uid1)
      have ph': phase s' CID = revPH using ph step unfolding a by (auto
simp: uu-defs)
      have ?match proof
      show validTrans ?trn1 using step1 by simp
      next
      show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi 1$  by
auto
      next
      show  $\gamma$  ?trn =  $\gamma$  ?trn1 using uidUIDs uid1UIDs unfolding a by
simp
      next
      assume  $\gamma$  ?trn thus g ?trn = g ?trn1 using uidUIDs unfolding a
by simp
      next
      have  $\Delta 2$  s' vl' s1' vl1 using ph' PID' s's1' unfolding  $\Delta 2$ -def vl'
apply(intro exI[of - CID] exI[of - uidc]) using l vl1-all
      using isPC-persistent[OF - step] revPH-pref-persists[OF rs PID -
step] ph
      not- $\varphi$ -isRev-isPC-persists[OF rs1 PID1  $\varphi 1$  step1]
      revPH-pref-persists[OF rs PID - step] not- $\varphi$ -isRev-isPC-persists[OF
rs PID  $\varphi$  step] ph uidc'
      by auto
      thus ? $\Delta$  s' vl' s1' vl1 by simp
      qed
      thus ?thesis by simp
      qed
      qed
      qed
      next
      case True note  $\varphi = \text{True}$ 
      then obtain cid uid p uid' where a: a = Cact (cReview cid uid p PID

```

$uid'$ )

**and**  $ou$ :  $ou = outOK$  **unfolding**  $\varphi$ -def2 **by** *auto*  
**have**  $cid$ :  $cid = CID$  **using** *step* **unfolding**  $a$   $ou$  **apply**(*auto simp: c-defs*)  
**by** (*metis PID paperIDs-equals rs*)  
**obtain**  $v$   $vl'$  **where**  $vl = v \# vl'$  **using**  $\varphi$   $c$  **unfolding** *consume-def* **by**  
(*cases vl*) *auto*  
**hence**  $vl$ :  $vl = v \# vl'$  **by** (*metis  $\varphi$  c consume-def list.sel(2-3)*)  
**obtain**  $v1$   $vl1'$  **where**  $vl1$ :  $vl1 = v1 \# vl1'$  **using**  $l$   $vl$  **by** (*cases vl1*) *auto*  
**obtain**  $uid1'$  **where**  $v1$ :  $v1 = (uid1', \{uid1'. isPC\ s\ CID\ uid1' \wedge pref\ s\ uid1'\ PID \neq Conflict\})$   
**using** *snd-isPC* **unfolding**  $vl1$  **by**(*cases v1*) *auto*  
**hence**  $uid1'$ :  $isPC\ s\ CID\ uid1' \wedge pref\ s\ uid1'\ PID \neq Conflict$  **and**  $uid1'1$ :  
 $\neg isRev\ s1\ CID\ uid1'\ PID$   
**using** *fst-isPC fst-isRev* **unfolding**  $vl1$  **by** *auto*  
**have**  $uid$ :  $isChair\ s\ CID\ uid \wedge pref\ s\ uid\ PID \neq Conflict$   
**using** *step* **unfolding**  $a$   $ou$   $cid$  **by** (*auto simp: c-defs*)  
**have**  $uid1$ :  $isChair\ s1\ CID\ uid \wedge pref\ s1\ uid\ PID \neq Conflict$   
**using**  $ss1$   $uid$  **unfolding** *eqExcPID-def* **using** *eqExcRLR-imp2* **by** *metis*  
**have**  $uid1'1$ :  $isPC\ s1\ CID\ uid1' \wedge pref\ s1\ uid1'\ PID \neq Conflict$   
**using**  $uid1'1$   $ss1$  **unfolding** *eqExcPID-def* **apply** *auto* **by** (*metis eqExcRLR-imp2*)  
**obtain**  $a1$  **where**  $a1$ :  $a1 = Cact\ (cReview\ CID\ uid\ p\ PID\ uid1')$  **by** *blast*  
**obtain**  $s1'$   $ou1$  **where**  $step1$ :  $step\ s1\ a1 = (ou1, s1')$  **by** (*metis prod.exhaust*)  
**have**  $ph1$ :  $phase\ s1\ CID = revPH$  **using**  $ss1$   $ph$  **unfolding** *eqExcPID-def*  
**by** *auto*  
**let**  $?trn1 = Trans\ s1\ a1\ ou1\ s1'$   
**have**  $s's1'$ :  $eqExcPID\ s'\ s1'$  **and**  $ou1$ :  $ou1 = outOK$   
**using** *eqExcPID-step- $\varphi$ -eqExcPID-out*[*OF rs rs1 a[unfolded cid]*  
 $a1\ ss1\ step[unfolded\ ou]\ uid1'\ uid1'1\ step1$ ]  
**by** *auto*  
**hence**  $many-s1'$ :  $PID \in \in paperIDs\ s1'\ CID\ isChair\ s1'\ CID\ uid \wedge pref\ s1'\ uid\ PID \neq Conflict$   
 $phase\ s1'\ CID = revPH\ pass\ s1'\ uid = pass\ s1\ uid$   
 $isPC\ s1'\ CID\ uid1' \wedge pref\ s1'\ uid1'\ PID \neq Conflict$   
**subgoal** **by** (*metis PID1 paperIDs-mono step1*)  
**subgoal** **by** (*metis (no-types, lifting) PID1 Suc-leI eqExcPID-def isChair-persistent lessI ph revPH-pref-persists rs1 ss1 step1 uid1*)  
**subgoal** **using**  $step1$   $ph1$  **unfolding**  $a1$  **by** (*fastforce simp: c-defs*)  
**subgoal** **using**  $step1$   $ph1$  **unfolding**  $a1$  **by** (*fastforce simp: c-defs*)  
**subgoal** **by** (*metis (no-types, lifting) PID1 Suc-leI eqExcPID-def isPC-persistent lessI ph revPH-pref-persists rs1 ss1 step1 uid1'*)  
**done**  
**hence**  $more-s1'$ :  $uid \in \in userIDs\ s1'\ CID \in \in confIDs\ s1'$   
**by** (*metis paperIDs-confIDs reach-PairI roles-userIDs rs1 step1 many-s1'(1)*)  
**have**  $\varphi1$ :  $\varphi\ ?trn1$  **unfolding**  $a1$   $ou1$   $\varphi$ -def2 **by** *auto*  
**have**  $f1$ :  $f\ ?trn1 = v1$  **unfolding**  $a1$   $v1$  **apply** *simp*  
**using**  $ss1$  **unfolding** *eqExcPID-def* **using** *eqExcRLR-imp2*  
**by** (*metis eqExcRLR-set isRevRoleFor.simps(3)*)

```

    have rs1': reach s1' using rs1 step1 by (auto intro: reach-PairI)
    have ph': phase s' CID = revPH using step ph unfolding a by (auto simp:
c-defs)
    have ?match proof
      show validTrans ?trn1 using step1 by simp
      next
        show consume ?trn1 vl1 vl1' unfolding consume-def vl1 using  $\varphi$ 1 f1
by auto
      next
        show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding a a1 by simp
      next
        assume  $\gamma$  ?trn thus g ?trn = g ?trn1 by (metis  $\varphi$  T- $\varphi$ - $\gamma$   $\gamma$ .simps rsT
step)
      next
        have 0: {uid'. isPC s' CID uid'  $\wedge$  pref s' uid' PID  $\neq$  Conflict} =
          {uid'. isPC s CID uid'  $\wedge$  pref s uid' PID  $\neq$  Conflict}
        using step rs ph unfolding a ou by (auto simp: c-defs)
        have 1: {uid'. isRev s1' CID uid' PID}  $\subseteq$  {uid'. isRev s1 CID uid' PID}
 $\cup$  {uid1'}
        using step1 unfolding a1 ou1 by (auto simp: c-defs isRev-def2)
        have 2: fst ' set vl1'  $\subseteq$  fst ' set vl1 - {uid1'} using dis
unfolding vl1 apply simp unfolding image-set unfolding v1 by simp
        have 3: fst ' set vl1'  $\cap$  {uid'. isRev s1' CID uid' PID} = {}
        using 1 2 vl1-all(3) by blast
        have  $\Delta$ 2 s' vl' s1' vl1' unfolding  $\Delta$ 2-def
        apply (intro exI[of - CID] exI[of - uidc]) using l vl1-all unfolding vl1
vl apply simp
        using PID' ph' uidc' s's1' apply simp
        unfolding 0 using 3 by simp
        thus ? $\Delta$  s' vl' s1' vl1' by simp
      qed
      thus ?thesis by simp
    qed
  qed
  thus ?thesis using l by (metis length-0-conv)
qed
qed

lemma unwind-cont- $\Delta$ 3: unwind-cont  $\Delta$ 3 { $\Delta$ 3, $\Delta$ e}
proof(rule, simp)
  let ? $\Delta$  =  $\lambda$ s vl s1 vl1.  $\Delta$ 3 s vl s1 vl1  $\vee$   $\Delta$ e s vl s1 vl1
  fix s s1 :: state and vl vl1 :: value list
  assume rsT: reachNT s and rs1: reach s1 and  $\Delta$ 3 s vl s1 vl1
  then obtain CID where rs: reach s and ph: phase s CID > revPH (is ?ph >
-)
  and PID: PID  $\in$  paperIDs s CID and ss1: eqExcPID s s1
  and vl1: vl1 = []
  using reachNT-reach unfolding  $\Delta$ 3-def by auto
  have PID1: PID  $\in$  paperIDs s1 CID

```

```

by (metis PID eqExcPID-sym isAut-paperIDs notInPaperIDs-eqExcPID-roles-eq
paperID-ex-userID rs rs1 ss1)
show iaction ? $\Delta$  s vl s1 vl1  $\vee$ 
  ((vl = []  $\longrightarrow$  vl1 = []))  $\wedge$  reaction ? $\Delta$  s vl s1 vl1 (is ?iact  $\vee$  (-  $\wedge$  ?react))
proof-
  have ?react
  proof
    fix a :: act and ou :: out and s' :: state and vl'
    let ?trn = Trans s a ou s'
    let ?ph' = phase s' CID
    assume step: step s a = (ou, s') and T:  $\neg$  T ?trn and c: consume ?trn vl vl'
    have PID': PID  $\in$  paperIDs s' CID using PID rs by (metis paperIDs-mono
step)
    have ph': phase s' CID > revPH using ph rs
    by (meson less-le-trans local.step phase-increases)
    have  $\varphi$ :  $\neg$   $\varphi$  ?trn using step ph unfolding  $\varphi$ -def2 apply(auto simp: c-defs)
    using PID paperIDs-equals rs by force
    have vl': vl' = vl using c  $\varphi$  unfolding consume-def by (cases vl) auto
    show match ? $\Delta$  s s1 vl1 a ou s' vl'  $\vee$  ignore ? $\Delta$  s s1 vl1 a ou s' vl' (is ?match
 $\vee$  ?ignore)
    proof(cases  $\chi$  a)
      case True
      thus ?thesis unfolding  $\chi$ -def2 proof(elim exE disjE)
        fix cid uid p n rc assume a: a = Uact (uReview cid uid p PID n rc)
        show ?thesis
        proof(cases ou = outErr)
          case True note ou = True
          hence s's: s' = s by (metis step step-outErr-eq)
          show ?thesis proof(cases uid  $\in$  UIDs)
            case True note uidUIDs = True
            obtain ou1 and s1' where step1: step s1 a = (ou1, s1') by (metis
prod.exhaust)
            let ?trn1 = Trans s1 a ou1 s1'
            have isPC s CID uid  $\longrightarrow$  pref s uid PID = Conflict
            using reachNT-non-isPC-isChair[OF rsT uidUIDs] ph PID by force
            hence 1: isPC s1 CID uid  $\longrightarrow$  pref s1 uid PID = Conflict using ss1
unfolding eqExcPID-def
            using eqExcRLR-imp2 by fastforce
            have ou1: ou1 = outErr using step1 uidUIDs unfolding a apply(auto
simp: u-defs)
            apply(cases cid = CID, auto)
            apply (metis 1 isRev-isPC isRev-pref-notConflict rs1)
            by (metis rs1 PID1 paperIDs-equals)
            have s1's1: s1' = s1 by (metis ou ou1 step step1 step-outErr-eq)
            have  $\varphi$ 1:  $\neg$   $\varphi$  ?trn1 using  $\varphi$  unfolding eqExcPID-step- $\varphi$ [OF rs rs1
ss1 PID ph step step1] .
            have ?match proof
              show validTrans ?trn1 using step1 by simp
            next

```

```

      show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi 1$  by
auto
    next
      show  $\gamma ?trn = \gamma ?trn1$  unfolding ss1 by simp
    next
      assume  $\gamma ?trn$  thus  $g ?trn = g ?trn1$  unfolding ou ou1 by simp
    next
      have  $\Delta 3 s' vl' s1' vl1$  using  $ph' PID' ss1$  unfolding  $\Delta 3$ -def  $s's$ 
 $s1's1 vl1$  by auto
      thus  $? \Delta s' vl' s1' vl1$  by simp
      qed
      thus ?thesis by simp
    next
      case False note uidUIDs = False
      have ?ignore proof
        show  $\neg \gamma ?trn$  using uidUIDs unfolding a by auto
      next
        have  $\Delta 3 s' vl' s1 vl1$  using  $ph' PID' ss1$  unfolding  $\Delta 3$ -def  $s's vl1$ 
by auto
        thus  $? \Delta s' vl' s1 vl1$  by simp
        qed
        thus ?thesis by simp
      qed
    next
      case False hence ou:  $ou = outOK$  using step unfolding a by (auto
simp: u-defs)
      obtain  $s1' uid1 p$  where
         $uid1: isRevNth s1 cid uid1 PID n$ 
        and  $step1: step s1 (Uact (uReview cid uid1 p PID n rc)) = (outOK, s1')$ 
(is step - ?a1 = -)
        and  $s's1': eqExcPID s' s1' using eqExcPID-step- $\chi 1$  rs rs1 a ss1 step$ 
ou by metis
      let  $?trn1 = Trans s1 ?a1 outOK s1'$ 
      have  $\varphi 1: \neg \varphi ?trn1$  by simp
      have  $isPC s cid uid \wedge pref s uid PID \neq Conflict$  using step unfolding
a ou apply(auto simp: u-defs)
      by (metis isRev-pref-notConflict-isPC rs)+
      hence uidUIDs:  $\neg uid \in UIDs$  using  $ph reachNT$ -non-isPC-isChair[OF
rsT] apply auto
      by (metis (no-types, opaque-lifting) eqExcPID-def isRevNth-geq-revPH
isRevNth-paperIDs not-le rs1 ss1 uid1)
      have  $isPC s1 cid uid1 \wedge pref s1 uid1 PID \neq Conflict$  using step1
apply(auto simp: u-defs)
      by (metis isRev-pref-notConflict-isPC rs1)+
      hence  $isPC s cid uid1 \wedge pref s uid1 PID \neq Conflict$  using ss1 unfolding
eqExcPID-def
      using eqExcRLR-imp2 by fastforce
      hence uid1UIDs:  $\neg uid1 \in UIDs$  using  $ph reachNT$ -non-isPC-isChair[OF
rsT] apply auto

```

```

      by (metis (no-types, opaque-lifting) eqExcPID-def isRevNth-geq-revPH
isRevNth-paperIDs not-le rs1 ss1 uid1)
      have ?match proof
        show validTrans ?trn1 using step1 by simp
      next
        show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi1$  by
auto
      next
        show  $\gamma$  ?trn =  $\gamma$  ?trn1 using uidUIDs uid1UIDs unfolding a by
simp
      next
        assume  $\gamma$  ?trn thus  $g$  ?trn =  $g$  ?trn1 using uidUIDs unfolding a
by simp
      next
        have  $\Delta3$  s' vl' s1' vl1 using ph' PID' s's1' unfolding  $\Delta3$ -def vl1 by
auto
        thus ? $\Delta$  s' vl' s1' vl1 by simp
        qed
        thus ?thesis by simp
      qed
    next
    fix cid uid p n rc assume a: a = UAct (uuReview cid uid p PID n rc)
    show ?thesis
    proof(cases ou = outErr)
      case True note ou = True
      hence s's: s' = s by (metis step step-outErr-eq)
      show ?thesis proof(cases uid  $\in$  UIDs)
        case True note uidUIDs = True
        obtain ou1 and s1' where step1: step s1 a = (ou1, s1') by (metis
prod.exhaust)
        let ?trn1 = Trans s1 a ou1 s1'
        have isPC s CID uid  $\longrightarrow$  pref s uid PID = Conflict
        using reachNT-non-isPC-isChair[OF rsT uidUIDs] ph PID by force
        hence 1: isPC s1 CID uid  $\longrightarrow$  pref s1 uid PID = Conflict using ss1
unfolding eqExcPID-def
        using eqExcRLR-imp2 by fastforce
        have ou1: ou1 = outErr using step1 uidUIDs unfolding a apply(auto
simp: uu-defs)
        apply(cases cid = CID, auto)
        apply (metis 1 isRev-isPC isRev-pref-notConflict rs1)
        by (metis rs1 PID1 paperIDs-equals)
        have s1's1: s1' = s1 by (metis ou ou1 step step1 step-outErr-eq)
        have  $\varphi1$ :  $\neg \varphi$  ?trn1 using  $\varphi$  unfolding eqExcPID-step- $\varphi$ [OF rs rs1
ss1 PID ph step step1] .
        have ?match proof
          show validTrans ?trn1 using step1 by simp
        next
          show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi1$  by
auto

```

```

next
  show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
next
  assume  $\gamma$  ?trn thus g ?trn = g ?trn1 unfolding ou ou1 by simp
next
  have  $\Delta\beta$  s' vl' s1' vl1 using ph' PID' ss1 unfolding  $\Delta\beta$ -def s's
s1's1 vl1 by auto
  thus ? $\Delta$  s' vl' s1' vl1 by simp
qed
thus ?thesis by simp
next
case False note uidUIDs = False
have ?ignore proof
  show  $\neg \gamma$  ?trn using uidUIDs unfolding a by auto
next
  have  $\Delta\beta$  s' vl' s1 vl1 using ph' PID' ss1 unfolding  $\Delta\beta$ -def s's vl1
by auto
  thus ? $\Delta$  s' vl' s1 vl1 by simp
qed
thus ?thesis by simp
qed
next
case False hence ou: ou = outOK using step unfolding a by (auto
simp: u-defs)
  obtain s1' uid1 p where
    uid1: isRevNth s1 cid uid1 PID n
  and step1: step s1 (UUact (uuReview cid uid1 p PID n rc)) = (outOK, s1')
(is step - ?a1 = -)
  and s's1': eqExcPID s' s1' using eqExcPID-step- $\chi^2$  rs rs1 a ss1 step
ou by metis
  let ?trn1 = Trans s1 ?a1 outOK s1'
  have  $\varphi$ 1:  $\neg \varphi$  ?trn1 by simp
  have isPC s cid uid  $\wedge$  pref s uid PID  $\neq$  Conflict using step unfolding
a ou apply(auto simp: uu-defs)
  by (metis isRev-pref-notConflict-isPC rs)+
  hence uidUIDs:  $\neg uid \in$  UIDs using ph reachNT-non-isPC-isChair[OF
rsT] apply auto
  by (metis (no-types, opaque-lifting) eqExcPID-def isRevNth-geq-revPH
isRevNth-paperIDs not-le rs1 ss1 uid1)
  have isPC s1 cid uid1  $\wedge$  pref s1 uid1 PID  $\neq$  Conflict using step1
apply(auto simp: uu-defs)
  by (metis isRev-pref-notConflict-isPC rs1)+
  hence isPC s cid uid1  $\wedge$  pref s uid1 PID  $\neq$  Conflict using ss1 unfolding
eqExcPID-def
  using eqExcRLR-imp2 by fastforce
  hence uid1UIDs:  $\neg uid1 \in$  UIDs using ph reachNT-non-isPC-isChair[OF
rsT] apply auto
  by (metis (no-types, opaque-lifting) eqExcPID-def isRevNth-geq-revPH
isRevNth-paperIDs not-le rs1 ss1 uid1)

```

```

      have ?match proof
        show validTrans ?trn1 using step1 by simp
      next
        show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi 1$  by
auto
      next
        show  $\gamma ?trn = \gamma ?trn1$  using uidUIDs uid1UIDs unfolding a by
simp
      next
        assume  $\gamma ?trn$  thus  $g ?trn = g ?trn1$  using uidUIDs unfolding a
by simp
      next
        have  $\Delta 3 s' vl' s1' vl1$  using  $ph' PID' s's1'$  unfolding  $\Delta 3$ -def vl1 by
auto
        thus ? $\Delta s' vl' s1' vl1$  by simp
        qed
        thus ?thesis by simp
      qed
    qed
  next
    case False note  $\chi = False$ 
    obtain ou1 and s1' where step1: step s1 a = (ou1, s1') by (metis
prod.exhaust)
    let ?trn1 = Trans s1 a ou1 s1'
    have  $\varphi 1: \neg \varphi ?trn1$  using  $\varphi$  unfolding eqExcPID-step- $\varphi[OF rs rs1 ss1$ 
PID ph step step1] .
    have out: userOfA a  $\in$  UIDs  $\longrightarrow$  ou1 = ou
    using eqExcPID-step-out[OF ss1 step step1 rsT rs1 PID -  $\varphi \varphi 1 \chi$ ] ph by
auto
    have s's1': eqExcPID s' s1' using eqExcPID-step rs rs1 ss1 step step1 PID
ph  $\varphi \chi$  by blast
    have ?match proof
      show validTrans ?trn1 using step1 by simp
    next
      show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi 1$  by auto
    next
      show  $\gamma ?trn = \gamma ?trn1$  unfolding ss1 by simp
    next
      assume  $\gamma ?trn$  thus  $g ?trn = g ?trn1$  using out by simp
    next
      have  $\Delta 3 s' vl' s1' vl1$  using  $ph' PID' s's1'$  unfolding  $\Delta 3$ -def vl1 by
auto
      thus ? $\Delta s' vl' s1' vl1$  by simp
      qed
      thus ?thesis by simp
    qed
  qed
  thus ?thesis using vl1 by simp
qed

```

qed

**definition** *K1exit* where

$$K1exit\ cid\ s \equiv PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid \geq revPH \wedge \\ \neg (\exists\ uid.\ isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict)$$

**lemma** *invarNT-K1exit*: *invarNT* (*K1exit cid*)

**unfolding** *invarNT-def* **apply** (*safe dest!*: *reachNT-reach*)

**subgoal for** - *a* **apply**(*cases a*)

**subgoal for** *x1* **apply**(*cases x1*) **apply** (*fastforce simp add: c-defs K1exit-def geq-noPH-confIDs*)<sup>+</sup> .

**subgoal for** *x2* **apply**(*cases x2*) **apply** (*auto simp add: u-defs K1exit-def paperIDs-equals*)

**apply** (*metis less-eq-Suc-le less-not-refl paperIDs-equals*) .

**subgoal for** *x3* **apply**(*cases x3*) **apply** (*auto simp add: uu-defs K1exit-def*) .

**by** *auto*

**done**

**lemma** *noVal-K1exit*: *noVal* (*K1exit cid*) *v*

**apply**(*rule noφ-noVal*)

**unfolding** *noφ-def* **apply** *safe*

**subgoal for** - *a* **apply**(*cases a*)

**subgoal for** *x1* **apply**(*cases x1*)

**apply** (*auto simp add: c-defs K1exit-def*)

**by** (*metis paperIDs-equals reachNT-reach*)

**by** *auto*

**done**

**definition** *K2exit* where

$$K2exit\ cid\ s\ v \equiv$$
$$PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid \geq revPH \wedge$$
$$snd\ v \neq \{uid'.\ isPC\ s\ cid\ uid' \wedge pref\ s\ uid'\ PID \neq Conflict\}$$

**lemma** *revPH-isPC-constant*:

**assumes** *s*: *reach s*

**and** *step s a = (ou, s')*

**and** *pid*  $\in \in paperIDs\ s\ cid$  **and** *phase s cid*  $\geq revPH$

**shows** *isPC s' cid uid' = isPC s cid uid'*

**using** *assms* **apply**(*cases a*)

**subgoal for** *x1* **apply**(*cases x1*) **apply** (*auto simp add: c-defs*)

**apply** (*metis paperIDs-confIDs*) .

**subgoal for** *x2* **apply**(*cases x2*) **apply** (*auto simp add: u-defs*) .

**subgoal for** *x3* **apply**(*cases x3*) **apply** (*auto simp add: uu-defs*) .

**by** *auto*

**lemma** *revPH-pref-constant*:

**assumes** *s*: *reach s*

**and**  $step\ s\ a = (ou, s')$   
**and**  $pid \in \in paperIDs\ s\ cid$  **and**  $phase\ s\ cid \geq revPH$   
**shows**  $pref\ s'\ uid\ pid = pref\ s\ uid\ pid$   
**using**  $assms$  **apply**( $cases\ a$ )  
  **subgoal for**  $x1$  **apply**( $cases\ x1$ ) **apply** ( $auto\ simp\ add: c-defs$ )  
    **apply** ( $metis\ paperIDs-getAllPaperIDs$ )  
    **apply** ( $metis\ Suc-n-not-le-n\ le-SucI\ paperIDs-equals$ )  
    **apply** ( $metis\ Suc-n-not-le-n\ le-SucI\ paperIDs-equals$ ) .  
  **subgoal for**  $x2$  **apply**( $cases\ x2$ ) **apply** ( $auto\ simp\ add: u-defs$ )  
    **apply** ( $metis\ Suc-n-not-le-n\ paperIDs-equals$ ) .  
  **subgoal for**  $x3$  **apply**( $cases\ x3$ ) **apply** ( $auto\ simp\ add: uu-defs$ ) .  
  **by**  $auto$

**lemma**  $invarNT-K2exit: invarNT\ (\lambda\ s.\ K2exit\ cid\ s\ v)$   
**unfolding**  $invarNT-def$  **apply** ( $safe\ dest!: reachNT-reach$ )  
**unfolding**  $K2exit-def$   
**by** ( $smt\ (verit)\ Collect-cong\ le-trans\ paperIDs-mono\ phase-increases\ revPH-isPC-constant\ revPH-pref-constant$ )

**lemma**  $noVal-K2exit: noVal2\ (K2exit\ cid)\ v$   
**unfolding**  $noVal2-def$  **apply**  $safe$   
  **subgoal for**  $- a$  **apply**( $cases\ a$ )  
    **subgoal for**  $x1$  **apply**( $cases\ x1$ )  
      **apply** ( $auto\ simp\ add: c-defs\ K2exit-def$ )  
      **apply** ( $metis\ paperIDs-equals\ reachNT-reach$ )+ .  
    **by**  $auto$   
  **done**

**definition**  $K3exit$  **where**  
 $K3exit\ cid\ s \equiv PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > revPH$

**lemma**  $invarNT-K3exit: invarNT\ (K3exit\ cid)$   
**unfolding**  $invarNT-def$  **apply** ( $safe\ dest!: reachNT-reach$ )  
  **subgoal for**  $- a$  **apply**( $cases\ a$ )  
    **subgoal for**  $x1$  **apply**( $cases\ x1$ ) **apply** ( $auto\ simp\ add: c-defs\ K3exit-def$ ) .  
    **subgoal for**  $x2$  **apply**( $cases\ x2$ ) **apply** ( $auto\ simp\ add: u-defs\ K3exit-def$ ) .  
    **subgoal for**  $x3$  **apply**( $cases\ x3$ ) **apply** ( $auto\ simp\ add: uu-defs\ K3exit-def$ ) .  
  **by**  $auto$   
  **done**

**lemma**  $noVal-K3exit: noVal\ (K3exit\ cid)\ v$   
**apply**( $rule\ no\varphi-noVal$ )  
**unfolding**  $no\varphi-def$  **apply**  $safe$   
  **subgoal for**  $- a$  **apply**( $cases\ a$ )  
    **subgoal for**  $x1$  **apply**( $cases\ x1$ )  
      **apply** ( $auto\ simp\ add: c-defs\ K3exit-def$ )  
      **using**  $reachNT-reach\ paperIDs-equals$  **by**  $fastforce$

```

    by auto
  done

lemma unwind-exit- $\Delta e$ : unwind-exit  $\Delta e$ 
proof
  fix s s1 :: state and vl vl1 :: value list
  assume rsT: reachNT s and rs1: reach s1 and  $\Delta e$ :  $\Delta e$  s vl s1 vl1
  hence vl: vl  $\neq$  [] using reachNT-reach unfolding  $\Delta e$ -def by auto
  then obtain CID where K1exit CID s  $\vee$  K2exit CID s (hd vl)  $\vee$  K3exit CID s
  using  $\Delta e$  unfolding K1exit-def K2exit-def K3exit-def  $\Delta e$ -def by auto
  thus vl  $\neq$  []  $\wedge$  exit s (hd vl) apply (simp add: vl)
  by (metis exitI2 exitI2-noVal2 invarNT-K1exit invarNT-K2exit invarNT-K3exit
      noVal-K1exit noVal-K2exit noVal-K3exit rsT)
qed

theorem secure: secure
apply (rule unwind-decomp3-secure[of  $\Delta 1$   $\Delta 2$   $\Delta e$   $\Delta 3$ ])
using
  istate- $\Delta 1$ 
  unwind-cont- $\Delta 1$  unwind-cont- $\Delta 2$  unwind-cont- $\Delta 3$ 
  unwind-exit- $\Delta e$ 
by auto

end
theory Reviewer-Assignment-NCPC-Aut
imports ../Observation-Setup Reviewer-Assignment-Value-Setup Bounded-Deducibility-Security.Composition
begin

```

## 9.4 Confidentiality protection from users who are not PC members or authors of the paper

We verify the following property:

A group of users UIDs learn nothing about the reviewers assigned to a paper PID except for the fact that they are PC members having no conflict with that paper unless/until one of the following occurs:

- the user becomes a PC member in the paper's conference having no conflict with that paper and the conference moves to the reviewing phase, or
- the user becomes an author of the paper and the conference moves to the notification phase.

```

fun T :: (state,act,out) trans  $\Rightarrow$  bool where
  T (Trans - - ou s') =
    ( $\exists$  uid  $\in$  UIDs.

```

$$\begin{aligned}
& (\exists \text{ cid. } PID \in \text{paperIDs } s' \text{ cid} \wedge \text{isPC } s' \text{ cid uid} \wedge \text{pref } s' \text{ uid } PID \neq \text{Conflict} \\
& \wedge \text{phase } s' \text{ cid} \geq \text{revPH}) \\
& \quad \vee \\
& (\exists \text{ cid. } PID \in \text{paperIDs } s' \text{ cid} \wedge \text{isAut } s' \text{ cid uid } PID \wedge \text{phase } s' \text{ cid} \geq \\
& \text{notifPH}) \\
& )
\end{aligned}$$

**declare**  $T.\text{simps}$  [ $\text{simp del}$ ]

**definition**  $B :: \text{value list} \Rightarrow \text{value list} \Rightarrow \text{bool}$  **where**

$$\begin{aligned}
B \text{ vl vl1} & \equiv \\
\text{vl} & \neq [] \wedge \\
& \text{distinct } (\text{map fst vl1}) \wedge \text{fst } ' (\text{set vl1}) \subseteq \text{snd } (\text{hd vl}) \wedge \text{snd } ' (\text{set vl1}) = \{\text{snd } (\text{hd} \\
& \text{vl})\}
\end{aligned}$$

**interpretation**  $BD\text{-Security-IO}$  **where**

$\text{istate} = \text{istate}$  **and**  $\text{step} = \text{step}$  **and**  
 $\varphi = \varphi$  **and**  $f = f$  **and**  $\gamma = \gamma$  **and**  $g = g$  **and**  $T = T$  **and**  $B = B$   
**done**

**lemma**  $\text{reachNT-non-isPC-isChair}$ :

**assumes**  $\text{reachNT } s$  **and**  $\text{uid} \in \text{UIDs}$

**shows**

$$(PID \in \text{paperIDs } s \text{ cid} \wedge \text{isPC } s \text{ cid uid} \longrightarrow \text{pref } s \text{ uid } PID = \text{Conflict} \vee \text{phase } s \text{ cid} < \text{revPH})$$

$$\wedge \\
(PID \in \text{paperIDs } s \text{ cid} \wedge \text{isChair } s \text{ cid uid} \longrightarrow \text{pref } s \text{ uid } PID = \text{Conflict} \vee \text{phase } s \text{ cid} < \text{revPH})$$

$$\wedge \\
(PID \in \text{paperIDs } s \text{ cid} \wedge \text{isAut } s \text{ cid uid } PID \longrightarrow \text{phase } s \text{ cid} < \text{notifPH})$$

**using**  $\text{assms}$

**apply**  $\text{induct}$

**subgoal by** ( $\text{auto simp: istate-def}$ )

**subgoal apply**( $\text{intro conjI}$ )

**subgoal by** ( $\text{metis (no-types, lifting) } T.\text{simps not-le-imp-less trans.collapse}$ )

**subgoal by** ( $\text{metis (mono-tags, lifting) reachNT-reach } T.\text{simps isChair-isPC not-le-imp-less reach.Step trans.collapse}$ )

**subgoal by** ( $\text{metis } T.\text{simps not-le-imp-less trans.collapse}$ ) .

**done**

**lemma**  $T\text{-}\varphi\text{-}\gamma$ :

**assumes**  $1: \text{reachNT } s$  **and**  $2: \text{step } s a = (\text{ou}, s') \varphi (\text{Trans } s a \text{ ou } s')$

**shows**  $\neg \gamma (\text{Trans } s a \text{ ou } s')$

**using**  $\text{reachNT-non-isPC-isChair}[OF 1] 2$  **unfolding**  $T.\text{simps } \varphi\text{-def2}$

**by** ( $\text{fastforce simp: c-defs isRev-imp-isRevNth-getReviewIndex}$ )

**lemma** *T-φ-γ-stronger*:  
**assumes** *s*: *reach s* **and** *0*: *PID ∈∈ paperIDs s cid*  
**and** *2*: *step s a = (ou,s')* *φ (Trans s a ou s')*  
**and** *1*:  $\forall uid \in UIDs. isChair\ s\ cid\ uid \longrightarrow pref\ s\ uid\ PID = Conflict \vee phase\ s\ cid < revPH$   
**shows**  $\neg \gamma (Trans\ s\ a\ ou\ s')$   
**proof** –  
  **have**  $\neg (\exists uid \in UIDs. \exists cid. PID \in \in paperIDs\ s\ cid \wedge isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict \wedge phase\ s\ cid \geq revPH)$   
  **using** *0 1 s* **by** (*metis not-le paperIDs-equals*)  
  **thus** *?thesis using assms unfolding T.simps φ-def2 by (force simp add: c-defs)*  
**qed**

**lemma** *T-φ-γ-1*:  
**assumes** *s*: *reachNT s* **and** *s1*: *reach s1* **and** *PID*: *PID ∈∈ paperIDs s cid*  
**and** *ss1*: *eqExcPID2 s s1*  
**and** *step1*: *step s1 a = (ou1,s1')* **and** *φ1*: *φ (Trans s1 a ou1 s1')*  
**and** *φ*:  $\neg \varphi (Trans\ s\ a\ ou\ s')$   
**shows**  $\neg \gamma (Trans\ s1\ a\ ou1\ s1')$   
**proof** –  
  **have**  $\forall uid \in UIDs. isChair\ s\ cid\ uid \longrightarrow pref\ s\ uid\ PID = Conflict \vee phase\ s\ cid < revPH$   
  **using** *reachNT-non-isPC-isChair[OF s] PID* **by** *auto*  
  **hence** *1*:  $\forall uid \in UIDs. isChair\ s1\ cid\ uid \longrightarrow pref\ s1\ uid\ PID = Conflict \vee phase\ s1\ cid < revPH$   
  **using** *ss1 unfolding eqExcPID2-def using eqExcRLR-imp2 by fastforce*  
  **have** *PID1*: *PID ∈∈ paperIDs s1 cid using PID ss1 eqExcPID2-imp by auto*  
  **show** *?thesis apply(rule T-φ-γ-stronger[OF s1 PID1 step1 φ1]) using 1 by auto*  
**qed**

**lemma** *notInPaperIDs-eqExLRL-roles-eq*:  
**assumes** *s*: *reach s* **and** *s1*: *reach s1* **and** *PID*:  $\neg PID \in \in paperIDs\ s\ cid$   
**and** *eq*: *eqExcPID2 s s1*  
**shows** *roles s cid uid = roles s1 cid uid*  
**proof** –  
  **have**  $\neg PID \in \in paperIDs\ s1\ cid$  **using** *PID eq unfolding eqExcPID2-def by auto*  
  **hence**  $\neg isRev\ s\ cid\ uid\ PID \wedge \neg isRev\ s1\ cid\ uid\ PID$  **using** *s s1 PID* **by** (*metis isRev-paperIDs*)  
  **thus** *?thesis using eq unfolding eqExcPID2-def eqExcRLR-def*  
  **by** (*metis Bex-set-list-ex filter-True isRev-def*)  
**qed**

**lemma** *eqExcPID2-step-out*:  
**assumes** *ss1*: *eqExcPID2 s s1*  
**and** *step*: *step s a = (ou,s')* **and** *step1*: *step s1 a = (ou1,s1')*

```

and  $sT$ : reachNT  $s$  and  $s1$ : reach  $s1$ 
and  $PID$ :  $PID \in \text{paperIDs } s \text{ cid}$  and  $ph$ : phase  $s \text{ cid} \geq \text{revPH}$ 
and  $\varphi$ :  $\neg \varphi (\text{Trans } s \text{ a ou } s')$  and  $\varphi1$ :  $\neg \varphi (\text{Trans } s1 \text{ a ou1 } s1')$ 
and  $UIDs$ : userOfA  $a \in UIDs$ 
shows  $ou = ou1$ 
proof –
  note  $s = \text{reachNT-reach}[OF \ sT]$ 
  have  $s'$ : reach  $s'$  and  $s1'$ : reach  $s1'$  by (metis reach-PairI  $s \ s1 \ \text{step} \ \text{step1}$ ) +
  have  $s's1'$ : eqExcPID2  $s' \ s1'$  by (metis PID  $\varphi \ \text{eqExcPID2-step} \ ph \ s \ ss1 \ \text{step} \ \text{step1}$ )
  note  $Inv = \text{reachNT-non-isPC-isChair}[OF \ sT \ UIDs]$ 
  note  $eqs = \text{eqExcPID2-imp}[OF \ ss1]$ 
  note  $eqs' = \text{eqExcPID2-imp1}[OF \ ss1]$ 

  note  $\text{simps}[simp] = c\text{-defs } u\text{-defs } uu\text{-defs } r\text{-defs } l\text{-defs } \text{Paper-dest-conv } \text{eqExcPID2-def}$ 
  note  $\text{simps2}[simp] = \text{eqExcRLLR-imp}[of \ s \ \dots \ s1, \ OF \ s] \ \text{eqExcRLLR-imp}[of \ s' \ \dots \ s1' \ \dots \ s1']$ 
     $\text{eqExcRLLR-imp}[of \ s \ \dots \ s1'] \ \text{eqExcRLLR-imp}[of \ s' \ \dots \ s1]$ 
     $\text{eqExcRLLR-imp2}[of \ s \ \dots \ s1] \ \text{eqExcRLLR-imp2}[of \ s' \ \dots \ s1']$ 
     $\text{eqExcRLLR-imp2}[of \ s \ \dots \ s1'] \ \text{eqExcRLLR-imp2}[of \ s' \ \dots \ s1]$ 
     $\text{eqExcRLLR-set}[of \ (\text{roles } s \ \text{cid } \text{uid}) \ (\text{roles } s1 \ \text{cid } \text{uid}) \ \text{for } \text{cid } \text{uid}]$ 
     $\text{eqExcRLLR-set}[of \ (\text{roles } s \ \text{cid } \text{uid}) \ (\text{roles } s1' \ \text{cid } \text{uid}) \ \text{for } \text{cid } \text{uid}]$ 
     $\text{eqExcRLLR-set}[of \ (\text{roles } s' \ \text{cid } \text{uid}) \ (\text{roles } s1 \ \text{cid } \text{uid}) \ \text{for } \text{cid } \text{uid}]$ 
     $\text{eqExcRLLR-set}[of \ (\text{roles } s' \ \text{cid } \text{uid}) \ (\text{roles } s1' \ \text{cid } \text{uid}) \ \text{for } \text{cid } \text{uid}]$ 
     $\text{foo2 } \text{foo3}$ 
     $\text{eqExcRLLR-imp-isRevRole-imp}$ 

  {fix  $\text{cid } \text{uid } p \ \text{pid}$  assume  $a = \text{Ract } (rMyReview \ \text{cid } \text{uid } p \ \text{pid})$ 
  hence  $?thesis$ 
  using  $\text{step} \ \text{step1} \ eqs \ eqs' \ s \ s1 \ UIDs \ PID \ \varphi \ \varphi1 \ ph$ 
   $\text{paperIDs-equals}[OF \ s] \ Inv$ 
  apply (auto simp add: isRev-getRevRole getRevRole-Some)[]
  apply (metis eqExcPID2-imp' isRev-isPC not-less option.inject pref-Conflict-isRev
  role.distinct role.inject ss1
  isRev-isPC not-less pref-Conflict-isRev simps2(1) simps2(8) is-
  Rev-getRevRole getRevRole-Some)+
  done
  } note  $\text{this}[simp]$ 

  {fix  $\text{cid } \text{uid } p \ \text{pid}$  assume  $a = \text{Ract } (rFinalDec \ \text{cid } \text{uid } p \ \text{pid})$ 
  hence  $?thesis$ 
  apply (cases  $\text{pid} = PID$ )
  using  $\text{step} \ \text{step1} \ eqs \ eqs' \ s \ s1 \ UIDs \ PID \ \varphi \ \varphi1 \ ph$ 
   $\text{paperIDs-equals}[OF \ s] \ Inv$  using  $\text{eqExcPID-RDD}$  by fastforce+
  } note  $\text{this}[simp]$ 

show  $?thesis$ 
  using  $\text{step} \ \text{step1} \ eqs \ eqs' \ s \ s1 \ UIDs \ PID \ \varphi \ \varphi1 \ ph$ 

```

```

paperIDs-equals[OF s] Inv
apply(cases a)
  subgoal for x1 by (cases x1; auto)
  subgoal for x2 apply(cases x2)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal apply clarsimp
    subgoal by (metis eqExcPID2-imp' isRev-pref-notConflict-isPC nat-neq-iff
simps2(7) ss1)
      subgoal by (metis isRev-pref-notConflict-isPC nat-neq-iff simps2(1)) . .
    subgoal for x3 apply(cases x3)
      subgoal by auto
      subgoal by auto
      subgoal apply clarsimp
      subgoal by (metis isRev-pref-notConflict-isPC le-antisym less-imp-le-nat
n-not-Suc-n simps2(1) simps2(5))
        subgoal by (metis isRev-pref-notConflict-isPC le-antisym less-imp-le-nat
n-not-Suc-n simps2(1)) .
      subgoal by auto .
    subgoal for x4 apply(cases x4)
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by clarsimp (metis eqExcPID2-RDD ss1)
      subgoal apply clarsimp
        subgoal by (metis eqExcPID2-RDD ss1)
          subgoal by auto .
      subgoal by auto
      subgoal by auto
      subgoal by clarsimp (metis eqExcPID2-imp2 less-imp-le-nat not-less-eq-eq
ss1)
        subgoal by clarsimp (metis less-imp-le-nat not-less-eq-eq)
        subgoal by clarsimp (metis discussion.inject less-imp-le-nat not-less-eq-eq)
        subgoal by clarsimp (metis (mono-tags, lifting) Suc-le-lessD not-less-eq)
        subgoal by auto
        subgoal by clarsimp linarith .
    subgoal for x5 apply(cases x5)
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by clarsimp (metis (no-types, opaque-lifting) empty-iff list.set(1)
notInPaperIDs-eqExLRL-roles-eq notIsPC-eqExLRL-roles-eq
simps2(5) ss1)
        subgoal by auto

```

subgoal by auto  
 subgoal by auto  
 subgoal by auto  
 subgoal by auto  
 subgoal by auto  
 subgoal by clarsimp (smt (verit) Suc-le-lessD eqExcRLR-imp filter-cong  
 isRev-pref-notConflict-isPC not-less-eq)  
 subgoal by clarsimp (metis Suc-le-lessD eqExcPID2-imp' not-less-eq ss1) .  
 done  
 qed

**definition**  $\Delta 1 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 1\ s\ vl\ s1\ vl1 \equiv$   
 $(\forall\ cid.\ PID \in \in paperIDs\ s\ cid \longrightarrow phase\ s\ cid < revPH) \wedge s = s1$   
 $\wedge B\ vl\ vl1$

**definition**  $\Delta 2 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 2\ s\ vl\ s1\ vl1 \equiv$   
 $\exists\ cid\ uid.$   
 $PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid = revPH \wedge$   
 $isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict \wedge$   
 $eqExcPID2\ s\ s1 \wedge$   
 $distinct\ (map\ fst\ vl1) \wedge$   
 $fst\ ' (set\ vl1) \subseteq \{uid'. isPC\ s\ cid\ uid' \wedge pref\ s\ uid'\ PID \neq Conflict\} \wedge$   
 $fst\ ' (set\ vl1) \cap \{uid'. isRev\ s1\ cid\ uid'\ PID\} = \{\} \wedge$   
 $snd\ ' (set\ vl1) \subseteq \{\{uid'. isPC\ s\ cid\ uid' \wedge pref\ s\ uid'\ PID \neq Conflict\}\}$

**definition**  $\Delta 3 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta 3\ s\ vl\ s1\ vl1 \equiv$   
 $\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > revPH \wedge eqExcPID2\ s\ s1 \wedge vl1 =$   
 $\square$

**definition**  $\Delta e :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$  **where**  
 $\Delta e\ s\ vl\ s1\ vl1 \equiv$   
 $vl \neq \square \wedge$   
 $($   
 $(\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid \geq revPH \wedge$   
 $\neg (\exists\ uid.\ isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict))$   
 $\vee$   
 $(\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid \geq revPH \wedge$   
 $snd\ (hd\ vl) \neq \{uid'. isPC\ s\ cid\ uid' \wedge pref\ s\ uid'\ PID \neq Conflict\})$   
 $\vee$   
 $(\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > revPH)$   
 $)$

**lemma** *istate- $\Delta 1$* :  
**assumes**  $B: B\ vl\ vl1$   
**shows**  $\Delta 1\ istate\ vl\ istate\ vl1$   
**using**  $B$  **unfolding**  $\Delta 1$ -def  $B$ -def *istate-def* **by** auto

```

lemma unwind-cont-Δ1: unwind-cont Δ1 {Δ1,Δ2,Δe}
proof(rule, simp)
  let ?Δ = λs vl s1 vl1. Δ1 s vl s1 vl1 ∨ Δ2 s vl s1 vl1 ∨ Δe s vl s1 vl1
  fix s s1 :: state and vl vl1 :: value list
  assume rsT: reachNT s and rs1: reach s1 and Δ1 s vl s1 vl1
  hence rs: reach s and ss1: s1 = s and B: B vl vl1
  and c-d: distinct (map fst vl1) ∧ fst ' (set vl1) ⊆ snd (hd vl) ∧ snd ' (set vl1)
= {snd (hd vl)}
  and vl: vl ≠ []
  and PID-ph: ∧ cid. PID ∈∈ paperIDs s cid ⇒ phase s cid < revPH
  using reachNT-reach unfolding Δ1-def B-def by auto
  have rv: ∧ cid. ¬ (∃ uid'. isRev s cid uid' PID) using rs PID-ph
  by (metis isRev-geq-revPH isRev-paperIDs less-Suc-eq-le not-less-eq-eq)
  show iaction ?Δ s vl s1 vl1 ∨
    ((vl = [] → vl1 = []) ∧ reaction ?Δ s vl s1 vl1) (is ?iact ∨ (- ∧ ?react))
  proof-
    have ?react proof
      fix a :: act and ou :: out and s' :: state and vl'
      let ?trn = Trans s a ou s' let ?trn1 = Trans s1 a ou s'
      assume step: step s a = (ou, s') and T: ¬ T ?trn and c: consume ?trn vl vl'
      have φ: ¬ φ ?trn
      apply(cases a)
      subgoal for x1 apply(cases x1) using step PID-ph by (fastforce simp:
c-defs)+
      by simp-all
      hence vl': vl' = vl using c unfolding consume-def by auto
      show match ?Δ s s1 vl1 a ou s' vl' ∨ ignore ?Δ s s1 vl1 a ou s' vl' (is ?match
∨ ?ignore)
    proof-
      have ?match proof
        show validTrans ?trn1 unfolding ss1 using step by simp
        next
        show consume ?trn1 vl1 vl1 unfolding consume-def ss1 using φ by auto
        next
        show γ ?trn = γ ?trn1 unfolding ss1 by simp
        next
        assume γ ?trn thus g ?trn = g ?trn1 unfolding ss1 by simp
        next
        show ?Δ s' vl' s' vl1
        proof(cases ∃ cid. PID ∈∈ paperIDs s cid)
          case False note PID = False
          have PID-ph': ∧ cid. PID ∈∈ paperIDs s' cid ⇒ phase s' cid < revPH
using PID step rs
          apply(cases a)
          subgoal for - x1 apply(cases x1) apply(fastforce simp: c-defs)+ .
          subgoal for - x2 apply(cases x2) apply(fastforce simp: u-defs)+ .
          subgoal for - x3 apply(cases x3) apply(fastforce simp: uu-defs)+ .
          by auto

```

```

      hence  $\Delta 1$   $s'$   $vl'$   $s'$   $vl1$  unfolding  $\Delta 1$ -def B-def  $vl'$  using PID-ph' c-d  $vl$ 
by auto
      thus ?thesis by auto
    next
      case True
      then obtain CID where PID: PID  $\in\in$  paperIDs s CID by auto
      hence ph: phase s CID < revPH using PID-ph by auto
      have PID': PID  $\in\in$  paperIDs s' CID by (metis PID paperIDs-mono
step)
      show ?thesis
      proof(cases phase s' CID < revPH)
        case True note ph' = True
        hence  $\Delta 1$   $s'$   $vl'$   $s'$   $vl1$  unfolding  $\Delta 1$ -def B-def  $vl'$  using  $vl$  c-d ph'
PID' apply auto
        by (metis reach-PairI paperIDs-equals rs step)
        thus ?thesis by auto
      next
        case False
        hence ph-rv': phase s' CID = revPH  $\wedge$   $\neg$  ( $\exists$  uid'. isRev s' CID uid'
PID)
        using ph PID step rs rv
        apply(cases a)
        subgoal for x1 apply(cases x1) apply(fastforce simp: c-defs)+ .
        subgoal for x2 apply(cases x2) apply(fastforce simp: u-defs
isRev-def2)+ .
        subgoal for x3 apply(cases x3) apply(fastforce simp: uu-defs)+ .
        by auto
      show ?thesis
      proof(cases ( $\exists$  uid. isChair s' CID uid  $\wedge$  pref s' uid PID  $\neq$  Conflict)
 $\wedge$ 
      snd (hd  $vl$ ) = {uid'. isPC s' CID uid'  $\wedge$  pref s' uid' PID  $\neq$ 
Conflict})
        case True
        hence  $\Delta 2$   $s'$   $vl'$   $s'$   $vl1$ 
        unfolding  $\Delta 2$ -def B-def  $vl'$  using c-d ph-rv' PID' by auto
        thus ?thesis by auto
      next
        case False
        hence  $\Delta e$   $s'$   $vl'$   $s'$   $vl1$ 
        unfolding  $\Delta e$ -def  $vl'$  using  $vl$  c-d ph-rv' PID' by auto
        thus ?thesis by auto
      qed
    qed
  qed
  thus ?thesis by simp
  qed
  thus ?thesis using  $vl$  by auto

```

qed  
qed

**lemma** *unwind-cont- $\Delta 2$* : *unwind-cont*  $\Delta 2$   $\{\Delta 2, \Delta 3, \Delta e\}$

**proof**(*rule, simp*)

let  $?\Delta = \lambda s \text{ vl } s1 \text{ vl1}. \Delta 2 \text{ s vl s1 vl1} \vee \Delta 3 \text{ s vl s1 vl1} \vee \Delta e \text{ s vl s1 vl1}$

**fix**  $s \text{ s1} :: \text{state}$  **and**  $\text{vl } \text{vl1} :: \text{value list}$

**assume**  $\text{rsT}: \text{reachNT } s$  **and**  $\text{rs1}: \text{reach } s1$  **and**  $\Delta 2 \text{ s vl s1 vl1}$

**then obtain**  $\text{CID } \text{uid}$  **where**  $\text{uid}: \text{isChair } s \text{ CID } \text{uid} \wedge \text{pref } s \text{ uid } \text{PID} \neq \text{Conflict}$

**and**  $\text{rs}: \text{reach } s$  **and**  $\text{ph}: \text{phase } s \text{ CID} = \text{revPH } (\text{is } ?\text{ph} = -)$  **and**  $\text{ss1}: \text{eqExcPID2 } s \text{ s1}$

**and**  $\text{PID}: \text{PID} \in \in \text{paperIDs } s \text{ CID}$

**and**  $\text{dis}: \text{distinct } (\text{map } \text{fst } \text{vl1})$

**and**  $\text{fst-isPC}: \text{fst } ' (\text{set } \text{vl1}) \subseteq \{\text{uid}' . \text{isPC } s \text{ CID } \text{uid}' \wedge \text{pref } s \text{ uid}' \text{PID} \neq \text{Conflict}\}$

**and**  $\text{fst-isRev}: \text{fst } ' (\text{set } \text{vl1}) \cap \{\text{uid}' . \text{isRev } s1 \text{ CID } \text{uid}' \text{PID}\} = \{\}$

**and**  $\text{snd-isPC}: \text{snd } ' (\text{set } \text{vl1}) \subseteq \{\{\text{uid}' . \text{isPC } s \text{ CID } \text{uid}' \wedge \text{pref } s \text{ uid}' \text{PID} \neq \text{Conflict}\}\}$

**using** *reachNT-reach unfolding*  $\Delta 2\text{-def}$  **by** *auto*

**hence**  $\text{uid-notin}: \text{uid} \notin \text{UIDs}$

**using** *reachNT-non-isPC-isChair rsT* **by** *fastforce*

**note**  $\text{vl1-all} = \text{dis } \text{fst-isPC } \text{fst-isRev } \text{snd-isPC}$

**show**  $\text{iaction } ?\Delta \text{ s vl s1 vl1} \vee$

$((\text{vl} = [] \longrightarrow \text{vl1} = []) \wedge \text{reaction } ?\Delta \text{ s vl s1 vl1}) (\text{is } ?\text{iact} \vee (- \wedge ?\text{react}))$

**proof**(*cases vl1*)

**case** (*Cons v1 vl1'*) **note**  $\text{vl1} = \text{Cons}$

**obtain**  $\text{uid}'$  **where**  $\text{v1}: \text{v1} = (\text{uid}', \{\text{uid}' . \text{isPC } s \text{ CID } \text{uid}' \wedge \text{pref } s \text{ uid}' \text{PID} \neq \text{Conflict}\})$

**using** *snd-isPC unfolding vl1* **by**(*cases v1*) *auto*

**hence**  $\text{uid}' : \text{isPC } s \text{ CID } \text{uid}' \wedge \text{pref } s \text{ uid}' \text{PID} \neq \text{Conflict}$

**and**  $\text{uid}'1: \neg \text{isRev } s1 \text{ CID } \text{uid}' \text{PID}$

**using** *fst-isPC fst-isRev unfolding vl1* **by** *auto*

**have**  $\text{uid1}: \text{isChair } s1 \text{ CID } \text{uid} \wedge \text{pref } s1 \text{ uid } \text{PID} \neq \text{Conflict}$

**using**  $\text{ss1 } \text{uid}$  **unfolding** *eqExcPID2-def* **using** *eqExcRLR-imp2* **by** *metis*

**define**  $a1$  **where**  $a1 \equiv \text{Cact } (\text{cReview } \text{CID } \text{uid} (\text{pass } s \text{ uid}) \text{PID } \text{uid}')$

**obtain**  $s1' \text{ ou1}$  **where**  $\text{step1}: \text{step } s1 \text{ a1} = (\text{ou1}, s1')$  **by** (*metis prod.exhaust*)

**let**  $?trn1 = \text{Trans } s1 \text{ a1 } \text{ou1 } s1'$

**have**  $s1s1': \text{eqExcPID2 } s1 \text{ s1}'$  **using**  $a1\text{-def } \text{step1 } \text{cReview-step-eqExcPID2}$  **by** *blast*

**have**  $\text{ss1}': \text{eqExcPID2 } s \text{ s1}'$  **using** *eqExcPID2-trans[OF ss1 s1s1']* .

**hence**  $\text{many-s1}': \text{PID} \in \in \text{paperIDs } s1' \text{ CID } \text{isChair } s1' \text{ CID } \text{uid} \wedge \text{pref } s1' \text{ uid } \text{PID} \neq \text{Conflict}$

$\text{phase } s1' \text{ CID} = \text{revPH } \text{pass } s1' \text{ uid} = \text{pass } s \text{ uid}$

$\text{isPC } s1' \text{ CID } \text{uid}' \wedge \text{pref } s1' \text{ uid}' \text{PID} \neq \text{Conflict}$

**using**  $\text{uid}' \text{uid } \text{PID } \text{ph}$  **unfolding** *eqExcPID2-def* **using** *eqExcRLR-imp2* **apply** *auto* **by** *metis+*

**hence**  $\text{more-s1}': \text{uid} \in \in \text{userIDs } s1' \text{ CID} \in \in \text{confIDs } s1'$

**by** (*metis paperIDs-confIDs reach-PairI roles-userIDs rs1 step1 many-s1'(1)+*)

**have**  $\text{ou1}: \text{ou1} = \text{outOK}$  **using**  $\text{step1}$  **unfolding**  $a1\text{-def}$  **apply** (*simp add:*

```

c-defs)
  using more-s1' many-s1' uid'1 by (metis roles-userIDs rs1)
  have f: f ?trn1 = v1 unfolding a1-def v1 apply simp
  using ss1 unfolding eqExcPID2-def using eqExcRLR-imp2
  by (metis eqExcRLR-set isRevRoleFor.simps(3))
  have rs1': reach s1' using rs1 step1 by (auto intro: reach-PairI)
  have ?iact proof
    show step s1 a1 = (ou1,s1') by fact
  next
    show  $\varphi$ :  $\varphi$  ?trn1 using ou1 unfolding a1-def by simp
    thus consume ?trn1 vl1 vl1' using f unfolding consume-def vl1 by simp
  next
    show  $\neg \gamma$  ?trn1 by (simp add: a1-def uid-notin)
  next
    have {uid'. isRev s1' CID uid' PID}  $\subseteq$  insert uid' {uid'. isRev s1 CID uid'
PID}
    using step1 unfolding a1-def ou1 by (auto simp add: c-defs isRev-def2 )
    hence fst ' set vl1'  $\cap$  {uid'. isRev s1' CID uid' PID} = {}
    using fst-isRev dis unfolding vl1 v1 by auto
    hence  $\Delta 2$  s vl s1' vl1' unfolding  $\Delta 2$ -def
    using PID ph ss1' uid using vl1-all unfolding vl1 by auto
    thus ? $\Delta$  s vl s1' vl1' by simp
  qed
  thus ?thesis by auto
next
  case Nil note vl1 = Nil
  have ?react proof
    fix a :: act and ou :: out and s' :: state and vl'
    let ?trn = Trans s a ou s'
    let ?ph' = phase s' CID
    assume step: step s a = (ou, s') and T:  $\neg T$  ?trn and c: consume ?trn vl vl'
    have PID': PID  $\in \in$  paperIDs s' CID using PID rs by (metis paperIDs-mono
step)
    have uid': isChair s' CID uid  $\wedge$  pref s' uid PID  $\neq$  Conflict
    using uid step rs ph PID isChair-persistent revPH-pref-persists[OF rs PID ]
  by auto
  have all-vl1':
    fst ' (set vl1)  $\subseteq$  {uid'. isPC s' CID uid'  $\wedge$  pref s' uid' PID  $\neq$  Conflict}
  and snd ' (set vl1)  $\subseteq$  {{uid'. isPC s' CID uid'  $\wedge$  pref s' uid' PID  $\neq$  Conflict}}
  using vl1-all
  apply (metis (full-types) empty-subsetI image-empty set-empty vl1)
  by (metis (lifting, no-types) empty-set image-is-empty subset-insertI vl1)
  show match ? $\Delta$  s s1 vl1 a ou s' vl'  $\vee$  ignore ? $\Delta$  s s1 vl1 a ou s' vl' (is ?match
 $\vee$  ?ignore)
  proof(cases  $\varphi$  ?trn)
    case False note  $\varphi =$  False
    have vl: vl' = vl using c  $\varphi$  unfolding consume-def by (cases vl) auto
    obtain ou1 and s1' where step1: step s1 a = (ou1,s1') by (metis
prod.exhaust)

```

```

let ?trn1 = Trans s1 a ou1 s1'
  have s's1': eqExcPID2 s' s1' using eqExcPID2-step[OF rs ss1 step step1
PID] ph  $\varphi$  by auto
  show ?thesis
  proof(cases ou = outErr  $\wedge$   $\neg$   $\gamma$  ?trn)
    case True note ou = True[THEN conjunct1] and  $\gamma$  = True[THEN
conjunct2]
    have s': s' = s using step ou by (metis step-outErr-eq)
    have ?ignore proof
      show  $\neg$   $\gamma$  ?trn by fact
    next
      show ? $\Delta$  s' vl' s1 vl1
      proof(cases ?ph' = revPH)
        case True
          hence  $\Delta$ 2 s' vl' s1 vl1 using ss1 PID' uid' vl1-all unfolding  $\Delta$ 2-def
vl1 s' by auto
          thus ?thesis by auto
        next
          case False hence ph': ?ph' > revPH using ph rs step s' by blast
          show ?thesis
          proof(cases vl')
            case Nil
              hence  $\Delta$ 3 s' vl' s1 vl1 using ss1 PID' ph' unfolding  $\Delta$ 3-def vl1 s'
by auto
              thus ?thesis by auto
            next
              case Cons
                hence  $\Delta$ e s' vl' s1 vl1 using PID' ph' unfolding  $\Delta$ e-def by auto
                thus ?thesis by auto
              qed
            qed
          qed
          thus ?thesis by auto
        next
          case Cons
            hence  $\Delta$ e s' vl' s1 vl1 using PID' ph' unfolding  $\Delta$ e-def by auto
            thus ?thesis by auto
          qed
        qed
      qed
    qed
  thus ?thesis by auto
next
case False note ou- $\gamma$  = False
  have  $\varphi$ 1:  $\neg$   $\varphi$  ?trn1
  using non-eqExcPID2-step- $\varphi$ -imp[OF rs ss1 PID - step step1  $\varphi$ ]
T- $\varphi$ - $\gamma$ -1[OF rsT rs1 PID ss1 step1 -  $\varphi$ ] ou- $\gamma$  by auto
  have ?match
  proof
    show validTrans ?trn1 using step1 by simp
  next
    show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi$ 1 by auto
  next
    show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
  next
    assume  $\gamma$  ?trn thus g ?trn = g ?trn1
  using eqExcPID2-step-out[OF ss1 step step1 rsT rs1 PID -  $\varphi$   $\varphi$ 1] ph by
simp

```

```

next
  show ? $\Delta$  s' vl' s1' vl1
  proof(cases ?ph' = revPH)
    case True note ph' = True
      hence  $\Delta 2$  s' vl' s1' vl1 using PID' s's1' uid' ph' vl1-all unfolding
 $\Delta 2$ -def vl1 by auto
      thus ?thesis by auto
    next
      case False hence ph': ?ph' > revPH using ph rs step by (metis
antisym-conv2 phase-increases)
      show ?thesis
      proof(cases vl')
        case Nil
          hence  $\Delta 3$  s' vl' s1' vl1 using s's1' PID' ph' unfolding  $\Delta 3$ -def vl1
by auto
          thus ?thesis by auto
        next
          case Cons
            hence  $\Delta e$  s' vl' s1' vl1 using PID' ph' unfolding  $\Delta e$ -def by auto
            thus ?thesis by auto
          qed
        qed
      qed
      thus ?thesis by simp
    qed
  next
    case True note  $\varphi = True$ 
    have s's: eqExcPID2 s' s using eqExcPID2-sym using  $\varphi$ -step-eqExcPID2[OF
 $\varphi$  step] .
    have s's1: eqExcPID2 s' s1 using eqExcPID2-trans[OF s's ss1] .
    have ?ignore proof
      show  $\neg \gamma$  ?trn using T- $\varphi$ - $\gamma$   $\varphi$  rsT step by auto
    next
      show ? $\Delta$  s' vl' s1 vl1
      proof(cases ?ph' = revPH)
        case True
          hence  $\Delta 2$  s' vl' s1 vl1 using s's1 PID' uid' vl1-all unfolding  $\Delta 2$ -def
vl1 by auto
          thus ?thesis by auto
        next
          case False hence ph': ?ph' > revPH using ph rs step by (metis
antisym-conv2 phase-increases)
          show ?thesis
          proof(cases vl')
            case Nil
              hence  $\Delta 3$  s' vl' s1 vl1 using s's1 PID' ph' unfolding  $\Delta 3$ -def vl1 by
auto
              thus ?thesis by auto
            next

```

```

      case Cons
      hence  $\Delta e s' vl' s1 vl1$  using  $PID' ph'$  unfolding  $\Delta e$ -def by auto
      thus ?thesis by auto
    qed
  qed
  qed
  thus ?thesis by auto
  qed
  qed
  thus ?thesis using  $vl1$  by auto
  qed
qed

```

```

lemma unwind-cont- $\Delta 3$ : unwind-cont  $\Delta 3$  { $\Delta 3, \Delta e$ }
proof(rule, simp)
  let ? $\Delta$  =  $\lambda s vl s1 vl1. \Delta 3 s vl s1 vl1 \vee \Delta e s vl s1 vl1$ 
  fix  $s s1$  :: state and  $vl vl1$  :: value list
  assume  $rsT$ : reachNT  $s$  and  $rs1$ : reach  $s1$  and  $\Delta 3 s vl s1 vl1$ 
  then obtain CID where  $rs$ : reach  $s$  and  $ph$ : phase  $s$  CID > revPH (is ?ph >
-)
  and  $PID$ :  $PID \in \in$  paperIDs  $s$  CID and  $ss1$ : eqExcPID2  $s s1$ 
  and  $vl1$ :  $vl1 = []$ 
  using reachNT-reach unfolding  $\Delta 3$ -def by auto
  show iaction ? $\Delta s vl s1 vl1 \vee$ 
    (( $vl = [] \longrightarrow vl1 = []$ )  $\wedge$  reaction ? $\Delta s vl s1 vl1$ ) (is ?iact  $\vee$  ( $- \wedge$  ?react))
  proof-
    have ?react
    proof
      fix  $a$  :: act and  $ou$  :: out and  $s'$  :: state and  $vl'$ 
      let ?trn = Trans  $s a ou s'$ 
      let ?ph' = phase  $s'$  CID
      assume step: step  $s a = (ou, s')$  and  $T$ :  $\neg T$  ?trn and  $c$ : consume ?trn  $vl vl'$ 
      have  $PID'$ :  $PID \in \in$  paperIDs  $s'$  CID using  $PID$   $rs$  by (metis paperIDs-mono
step)
      have  $ph'$ : phase  $s'$  CID > revPH using  $ph$   $rs$  by (meson less-le-trans local.step
phase-increases)
      show match ? $\Delta s s1 vl1 a ou s' vl' \vee$  ignore ? $\Delta s s1 vl1 a ou s' vl'$  (is ?match
 $\vee$  ?ignore)
      proof(cases  $\varphi$  ?trn)
        case False note  $\varphi = False$ 
        have  $vl'$ :  $vl' = vl$  using  $c$   $\varphi$  unfolding consume-def by (cases  $vl$ ) auto
        obtain  $ou1$  and  $s1'$  where step1: step  $s1 a = (ou1, s1')$  by (metis
prod.exhaust)
        let ?trn1 = Trans  $s1 a ou1 s1'$ 
        have  $s's1'$ : eqExcPID2  $s' s1'$  using eqExcPID2-step[OF  $rs$   $ss1$  step step1
PID]  $ph$   $\varphi$  by auto
        have  $\varphi1$ :  $\neg \varphi$  ?trn1 using  $\varphi$  unfolding eqExcPID2-step- $\varphi$ [OF  $rs$   $rs1$   $ss1$ 
PID  $ph$  step step1] .

```

```

have ?match proof
  show validTrans ?trn1 using step1 by simp
next
  show consume ?trn1 vl1 vl1 unfolding consume-def using  $\varphi1$  by auto
next
  show  $\gamma$  ?trn =  $\gamma$  ?trn1 unfolding ss1 by simp
next
  assume  $\gamma$  ?trn thus  $g$  ?trn =  $g$  ?trn1
  using eqExcPID2-step-out[OF ss1 step step1 rsT rs1 PID -  $\varphi$   $\varphi1$ ] ph by
simp
next
  have  $\Delta3$  s' vl' s1' vl1 using ph' PID' s's1' unfolding  $\Delta3$ -def vl1 by
auto
  thus ? $\Delta$  s' vl' s1' vl1 by simp
qed
thus ?thesis by simp
next
  case True note  $\varphi = True$ 
  have s's: eqExcPID2 s' s using eqExcPID2-sym[OF  $\varphi$ -step-eqExcPID2[OF
 $\varphi$  step]] .
  have s's1: eqExcPID2 s' s1 using eqExcPID2-trans[OF s's ss1] .
  have ?ignore proof
    show  $\neg \gamma$  ?trn using T- $\varphi$ - $\gamma$   $\varphi$  rsT step by auto
  next
    have  $\Delta3$  s' vl' s1 vl1 using s's1 PID' ph' vl1 unfolding  $\Delta3$ -def by auto
    thus ? $\Delta$  s' vl' s1 vl1 by auto
  qed
  thus ?thesis by simp
qed
qed
thus ?thesis using vl1 by simp
qed
qed

```

**definition** *K1exit* where

$$K1exit\ cid\ s \equiv PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid \geq revPH \wedge \\ \neg (\exists\ uid.\ isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict)$$

**lemma** *invarNT-K1exit*: *invarNT* (*K1exit cid*)

```

unfolding invarNT-def
apply (safe dest!: reachNT-reach)
subgoal for - a apply(cases a)
  subgoal for x1 apply(cases x1) apply (fastforce simp add: c-defs K1exit-def
geq-noPH-confIDs)+ .
  subgoal for x2 apply(cases x2) apply (auto simp add: u-defs K1exit-def
paperIDs-equals)
  apply (metis less-eq-Suc-le less-not-refl paperIDs-equals) .

```

**subgoal for  $x3$  apply(cases  $x3$ ) apply (auto simp add: uu-defs K1exit-def) .**  
**by auto**  
**done**

**lemma noVal-K1exit: noVal (K1exit cid) v**  
**apply(rule no $\varphi$ -noVal)**  
**unfolding no $\varphi$ -def**  
**apply safe**  
**subgoal for - a apply(cases a)**  
**subgoal for  $x1$  apply(cases  $x1$ )**  
**apply (auto simp add: c-defs K1exit-def)**  
**by (metis paperIDs-equals reachNT-reach)**  
**by auto**  
**done**

**definition K2exit where**

$K2exit\ cid\ s\ v \equiv$   
 $PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid \geq revPH \wedge$   
 $snd\ v \neq \{uid'. isPC\ s\ cid\ uid' \wedge pref\ s\ uid'\ PID \neq Conflict\}$

**lemma revPH-isPC-constant:**

**assumes  $s: reach\ s$**   
**and step  $s\ a = (ou, s')$**   
**and  $pid \in \in paperIDs\ s\ cid$  and  $phase\ s\ cid \geq revPH$**   
**shows  $isPC\ s'\ cid\ uid' = isPC\ s\ cid\ uid'$**   
**using *assms***  
**apply(cases  $a$ )**  
**subgoal for  $x1$  apply(cases  $x1$ )**  
**apply (auto simp add: c-defs)**  
**by (metis paperIDs-confIDs)**  
**subgoal for  $x2$  apply(cases  $x2$ ) apply (auto simp add: u-defs) .**  
**subgoal for  $x3$  apply(cases  $x3$ ) apply (auto simp add: uu-defs) .**  
**by auto**

**lemma revPH-pref-constant:**

**assumes  $s: reach\ s$**   
**and step  $s\ a = (ou, s')$**   
**and  $pid \in \in paperIDs\ s\ cid$  and  $phase\ s\ cid \geq revPH$**   
**shows  $pref\ s'\ uid\ pid = pref\ s\ uid\ pid$**   
**using *assms***  
**apply(cases  $a$ )**  
**subgoal for  $x1$  apply(cases  $x1$ )**  
**apply (auto simp add: c-defs)**  
**apply (metis paperIDs-getAllPaperIDs)**  
**apply (metis Suc-n-not-le-n le-SucI paperIDs-equals)**  
**apply (metis Suc-n-not-le-n le-SucI paperIDs-equals) .**  
**subgoal for  $x2$  apply(cases  $x2$ )**  
**apply (auto simp add: u-defs)**  
**apply (metis Suc-n-not-le-n paperIDs-equals) .**

**subgoal for**  $x3$  **apply**(*cases*  $x3$ ) **apply** (*auto simp add: uu-defs*) .  
**by** *auto*

**lemma** *invarNT-K2exit*: *invarNT* ( $\lambda s. K2exit\ cid\ s\ v$ )  
**unfolding** *invarNT-def* **apply** (*safe dest!: reachNT-reach*)  
**unfolding** *K2exit-def*  
**by** (*smt (verit) Collect-cong le-trans paperIDs-mono phase-increases revPH-isPC-constant revPH-pref-constant*)

**lemma** *noVal-K2exit*: *noVal2* ( $K2exit\ cid$ )  $v$   
**unfolding** *noVal2-def*  
**apply** *safe*  
**subgoal for** -  $a$  **apply**(*cases*  $a$ )  
**subgoal for**  $x1$  **apply**(*cases*  $x1$ )  
**apply** (*auto simp add: c-defs K2exit-def*)  
**by** (*metis paperIDs-equals reachNT-reach*)+  
**by** *auto*  
**done**

**definition** *K3exit* **where**  
 $K3exit\ cid\ s \equiv PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > revPH$

**lemma** *invarNT-K3exit*: *invarNT* ( $K3exit\ cid$ )  
**unfolding** *invarNT-def*  
**apply** (*safe dest!: reachNT-reach*)  
**subgoal for** -  $a$  **apply**(*cases*  $a$ )  
**subgoal for**  $x1$  **apply**(*cases*  $x1$ ) **apply** (*auto simp add: c-defs K3exit-def*) .  
**subgoal for**  $x2$  **apply**(*cases*  $x2$ ) **apply** (*auto simp add: u-defs K3exit-def*) .  
**subgoal for**  $x3$  **apply**(*cases*  $x3$ ) **apply** (*auto simp add: uu-defs K3exit-def*) .  
**by** *auto*  
**done**

**lemma** *noVal-K3exit*: *noVal* ( $K3exit\ cid$ )  $v$   
**apply**(*rule noφ-noVal*)  
**unfolding** *noφ-def*  
**apply** *safe*  
**subgoal for** -  $a$  **apply**(*cases*  $a$ )  
**subgoal for**  $x1$  **apply**(*cases*  $x1$ )  
**apply** (*auto simp add: c-defs K3exit-def*)  
**using** *paperIDs-equals reachNT-reach* **by** *fastforce*  
**by** *auto*  
**done**

**lemma** *unwind-exit-Δe*: *unwind-exit*  $\Delta e$   
**proof**  
**fix**  $s\ s1 :: state$  **and**  $vl\ vl1 :: value\ list$   
**assume**  $rsT: reachNT\ s$  **and**  $rs1: reach\ s1$  **and**  $\Delta e: \Delta e\ s\ vl\ s1\ vl1$

**hence**  $vl \neq []$  **using** *reachNT-reach* **unfolding**  $\Delta e\text{-def}$  **by** *auto*  
**then obtain** *CID* **where**  $K1\text{exit } CID \ s \vee K2\text{exit } CID \ s \ (hd \ vl) \vee K3\text{exit } CID \ s$   
**using**  $\Delta e$  **unfolding**  $K1\text{exit-def } K2\text{exit-def } K3\text{exit-def } \Delta e\text{-def}$  **by** *auto*  
**thus**  $vl \neq [] \wedge \text{exit } s \ (hd \ vl)$  **apply** (*simp add: vl*)  
**by** (*metis exitI2 exitI2-noVal2 invarNT-K1exit invarNT-K2exit invarNT-K3exit*  
*noVal-K1exit noVal-K2exit noVal-K3exit rsT*)

**qed**

**theorem** *secure: secure*  
**apply** (*rule unwind-decomp3-secure*[*of*  $\Delta 1 \ \Delta 2 \ \Delta e \ \Delta 3$ ])  
**using**  
*istate- $\Delta 1$*   
*unwind-cont- $\Delta 1$  unwind-cont- $\Delta 2$  unwind-cont- $\Delta 3$*   
*unwind-exit- $\Delta e$*   
**by** *auto*

**end**  
**theory** *Reviewer-Assignment-All*  
**imports**  
*Reviewer-Assignment-NCPC*  
*Reviewer-Assignment-NCPC-Aut*  
**begin**

**end**  
**theory** *Traceback-Properties*  
**imports** *Safety-Properties*  
**begin**

## 10 Traceback properties

In this section, we prove various traceback properties, by essentially giving trace-based justifications of certain occurring situations that are relevant for access to information:

**Being an author.** If a user is an author of a paper, then either the user has registered the paper in the first place or, inductively, has been appointed as coauthor by another author.

**Being a chair.** If a user is a chair of a conference, then either that user has registered the conference which has been approved by the superuser or, inductively, that user has been appointed by an existing chair of that conference.

**Being a PC member.** If a user is a PC member in a conference, then the user either must have been the original chair or must have been appointed by a chair.

**Being a reviewer.** If a user is a paper’s reviewer, then the user must have been appointed by a chair (from among the PC members who have not declared a conflict with the paper).

**Having conflict.** If a user has conflict with a paper, then the user is either an author of the paper or the conflict has been declared by that user or by a paper’s author, in such a way that between the moment when the conflict has been last declared and the current moment there is no transition that successfully removes the conflict.

**Conference reaching a phase.** If a conference is in a given phase different from “no phase”, then this has happened as a consequence of either a conference approval action by the superuser (if the phase is Setup) or a phase change action by a chair (otherwise).

More details and explanations can be found in [6, Section 3.6].

## 10.1 Preliminaries

**inductive** *trace-between* :: *state*  $\Rightarrow$  (*state,act,out*) *trans trace*  $\Rightarrow$  *state*  $\Rightarrow$  *bool*  
**where**

*empty[simp]*: *trace-between* *s* [] *s*  
| *step*: [[*trace-between* *s tr sh*; *step sh a* = (*ou,s'*)]  $\Longrightarrow$  *trace-between* *s* (*tr@[Trans sh a ou s']*) *s'*

**inductive-simps**

*trace-ft-empty[simp]*: *trace-between* *s* [] *s'* **and**  
*trace-ft-snoc*: *trace-between* *s* (*tr@[trn]*) *s'*

**thm** *trace-ft-empty trace-ft-snoc*

**lemma** *trace-ft-append*: *trace-between* *s* (*tr1@tr2*) *s'*

$\longleftrightarrow$  ( $\exists$  *sh*. *trace-between* *s tr1 sh*  $\wedge$  *trace-between* *sh tr2 s'*)

**apply** (*induction tr2 arbitrary*: *s'* *rule*: *rev-induct*)

**apply** *simp*

**apply** (*subst append-assoc[symmetric]*, *subst trace-ft-snoc*)

**apply** (*auto simp*: *trace-ft-snoc*)

**done**

**lemma** *trace-ft-Cons*: *trace-between* *s* (*trn#tr*) *s'*

$\longleftrightarrow$  ( $\exists$  *sh ou a*. *trn* = *Trans s a ou sh*  $\wedge$  *step s a* = (*ou,sh*)  $\wedge$  *trace-between* *sh tr s'*)

**apply** (*subst trace-ft-append[where ?tr1.0 = [trn], simplified]*)

**apply** (*subst trace-ft-snoc[where tr = [], simplified]*)

**by** *auto*

**lemmas** *trace-ft-simps* = *trace-ft-empty trace-ft-snoc trace-ft-Cons trace-ft-append*

**inductive** *trace-to* :: (*state,act,out*) *trans trace*  $\Rightarrow$  *state*  $\Rightarrow$  *bool* **where**

*empty*:  $\text{trace-to } [] \text{ istate}$   
 $| \text{step: } \llbracket \text{trace-to } tr \ s; \text{ step } s \ a = (ou, s') \rrbracket \implies \text{trace-to } (tr@[Trans \ s \ a \ ou \ s']) \ s'$

**lemma** *trace-to-ft*:  $\text{trace-to } tr \ s \longleftrightarrow \text{trace-between } \text{istate } tr \ s$

**proof** (*rule,goal-cases*)

**case 1 thus** *?case*

**by** *induction (auto intro: trace-between.intros)*

**next**

**case 2**

**moreover**

{**fix** *s'* **assume** *trace-between s' tr s* **hence**  $s' = \text{istate} \longrightarrow \text{trace-to } tr \ s$

**by** *induction (auto intro: trace-to.intros)*

}

**ultimately show** *?case* **by** *auto*

**qed**

**inductive-simps** *trace-to-empty[simp]*:  $\text{trace-to } [] \ s$

**lemma** *trace-to-reach*: **assumes** *trace-to tr s* **shows** *reach s*

**using** *assms* **apply** *induction*

**apply** (*rule reach.intros*)

**by** (*metis reach-step snd-conv*)

**lemma** *reach-to-trace*: **assumes** *reach s* **obtains** *tr* **where** *trace-to tr s*

**using** *assms* **apply** (*induction rule: reach-step-induct*)

**apply** (*auto intro: trace-to.intros*) []

**by** (*metis surjective-pairing trace-to.step*)

**lemma** *reach-trace-to-conv*:  $\text{reach } s \longleftrightarrow (\exists tr. \text{trace-to } tr \ s)$

**by** (*blast intro: trace-to-reach elim: reach-to-trace*)

**thm** *trace-to.induct[no-vars]*

**lemma** *trace-to-induct[case-names empty step, induct set]*:

$\llbracket \text{trace-to } x1 \ x2; P \ [] \ \text{istate};$

$\bigwedge tr \ s \ a \ ou \ s'.$

$\llbracket \text{trace-to } tr \ s; P \ tr \ s; \text{reach } s; \text{reach } s'; \text{step } s \ a = (ou, s') \rrbracket$

$\implies P \ (tr \ \#\#\ \text{Trans } s \ a \ ou \ s') \ s'$

$\implies P \ x1 \ x2$

**apply** (*erule trace-to.induct*)

**apply** *simp*

**apply** (*frule trace-to-reach*)

**using** *reach-PairI* **by** *blast*

## 10.2 Authorship

Only the creator of a paper, and users explicitly added by other authors, are authors of a paper.

**inductive** *isAut'* ::  $(state, act, out) \ \text{trans } \text{trace} \Rightarrow \text{confID} \Rightarrow \text{userID} \Rightarrow \text{paperID} \Rightarrow$

```

bool where
  creator: [ trn = Trans - (Cact (cPaper cid uid - pid -)) outOK - ]
    ⇒ isAut' (tr@[trn]) cid uid pid

| co-author: [
  isAut' tr cid uid' pid;
  trn = Trans - (Cact (cAuthor cid uid' - pid uid)) outOK - ]
  ⇒ isAut' (tr@[trn]) cid uid pid

| irrelevant: isAut' tr cid uid' pid ⇒ isAut' (tr@[ - ]) cid uid' pid

lemma justify-author:
  assumes trace-to tr s
  assumes isAut s cid uid pid
  shows isAut' tr cid uid pid
  using assms
proof (induction arbitrary: uid)
  case (empty s) thus ?case
    by (auto simp add: istate-def)
next
  case (step tr s a ou s')
  show ?case
  proof (cases isAut s cid uid pid)
    case True with step.IH show ?thesis by (blast intro: isAut'.intros)
  next
    case False
    with step.hyps step.prem1 obtain
      pass s1 s2 uid' where
        a=Cact (cPaper cid uid pass pid s1 s2)
        ∨ (a=Cact (cAuthor cid uid' pass pid uid) ∧ isAut s cid uid' pid)
    and [simp]: ou=outOK
    apply (cases a)
    subgoal for x1 apply (cases x1, auto simp add: c-defs) [] .
    subgoal for x2 apply (cases x2, auto simp add: u-defs) [] .
    subgoal for x3 apply (cases x3, auto simp add: uu-defs) [] .
    by simp-all
  thus ?thesis using step.IH
  apply (elim disjE)
  apply (rule isAut'.creator, auto) []
  apply (rule isAut'.co-author, auto) []
  done
qed
qed

```

```

lemma author-justify:
  assumes trace-to tr s
  assumes isAut' tr cid uid pid
  shows isAut s cid uid pid

```

```

using assms
proof (induction arbitrary: uid)
  case (empty s) thus ?case
    by (auto simp add: istate-def elim: isAut'.cases)
next
  case (step tr s a ou s')
  from step.prems
  show ?case
  proof (cases)
    case (creator - - pass s1 s2)
    hence [simp]: a=Cact (cPaper cid uid pass pid s1 s2) ou=outOK by simp-all
    from step.hyps show ?thesis
    by (auto simp add: c-defs)
  next
    case (co-author - uid' - - pass)
    hence [simp]: a=Cact (cAuthor cid uid' pass pid uid) ou=outOK by simp-all
    from step.hyps show ?thesis
    by (auto simp add: c-defs)
  next
    case (irrelevant) with step.IH have AUT: isAut s cid uid pid by simp

  note roles-confIDs[OF ‹reach s› AUT]
  with AUT ‹step s a = (ou, s')› show ?thesis
  apply (cases a)
  subgoal for x1 apply (cases x1, auto simp: c-defs) [] .
  subgoal for x2 apply (cases x2, auto simp: u-defs) [] .
  subgoal for x3 apply (cases x3, auto simp: uu-defs) [] .
  by simp-all
qed
qed

```

**theorem** *isAut-eq: trace-to tr s  $\implies$  isAut s cid uid pid  $\iff$  isAut' tr cid uid pid*

```

using justify-author author-justify
by (blast)

```

### 10.3 Becoming a Conference Chair

```

inductive isChair' :: (state,act,out) trans trace  $\implies$  confID  $\implies$  userID  $\implies$  bool where
  creator: [] trn=Trans - (Cact (cConf cid uid - -)) outOK - []
     $\implies$  isChair' (tr@[trn]) cid uid
| add-chair: [] isChair' tr cid uid'; trn = Trans - (Cact (cChair cid uid' - uid))
  outOK - []
     $\implies$  isChair' (tr@[trn]) cid uid
| irrelevant: [] isChair' tr cid uid []  $\implies$  isChair' (tr@[ - ]) cid uid

```

```

lemma justify-chair:
assumes trace-to tr s
assumes isChair s cid uid

```

```

shows isChair' tr cid uid
using assms
proof (induction arbitrary: uid)
  case (empty s) thus ?case
    by (auto simp add: istate-def)
next
case (step tr s a ou s')
show ?case
proof (cases isChair s cid uid)
  case True with step.IH show ?thesis by (blast intro: isChair'.intros)
next
case False
term cConf
with step.hyps step.premis obtain
  pass s1 s2 uid' where
  a=Cact (cConf cid uid pass s1 s2)
  ∨ (a=Cact (cChair cid uid' pass uid) ∧ isChair s cid uid')
  and [simp]: ou=outOK
  apply (cases a)
  subgoal for x1 apply (cases x1, auto simp add: c-defs) [] .
  subgoal for x2 apply (cases x2, auto simp add: u-defs) [] .
  subgoal for x3 apply (cases x3, auto simp add: uu-defs) [] .
  by simp-all
thus ?thesis using step.IH
  apply (elim disjE)
  apply (rule isChair'.creator, auto) []
  apply (rule isChair'.add-chair, auto) []
  done
qed
qed

lemma chair-justify:
  assumes trace-to tr s
  assumes isChair' tr cid uid
  shows isChair s cid uid
  using assms
proof (induction arbitrary: uid)
  case (empty s) thus ?case
    by (auto simp add: istate-def elim: isChair'.cases)
next
case (step tr s a ou s')
from step.premis
show ?case
proof (cases)
  case (creator - - pass s1 s2)
  hence [simp]: a=Cact (cConf cid uid pass s1 s2) ou=outOK by simp-all
  from step.hyps show ?thesis
    by (auto simp add: c-defs)
next

```

```

    case (add-chair - uid' - - pass)
    hence [simp]: a=Cact (cChair cid uid' pass uid) ou=outOK by simp-all
    from step.hyps show ?thesis
      by (auto simp add: c-defs)
  next
    case (irrelevant) with step.IH have CH: isChair s cid uid by simp

    from CH ⟨step s a = (ou, s')⟩ show ?thesis
      apply (cases a)
      subgoal for x1 apply (cases x1, auto simp: c-defs) [] .
      subgoal for x2 apply (cases x2, auto simp: u-defs) [] .
      subgoal for x3 apply (cases x3, auto simp: uu-defs) [] .
      by simp-all
  qed
qed

```

```

theorem isChair-eq: trace-to tr s  $\implies$  isChair s cid uid = isChair' tr cid uid

using justify-chair chair-justify
by (blast)

```

## 10.4 Committee Membership

```

inductive isPC' :: (state,act,out) trans trace  $\implies$  confID  $\implies$  userID  $\implies$  bool where
  chair: isChair' tr cid uid  $\implies$  isPC' tr cid uid
| add-com: [ isChair' tr cid uid'; trn = Trans - (Cact (cPC cid uid' - uid)) outOK
- ]
 $\implies$  isPC' (tr@[trn]) cid uid
| irrelevant: [isPC' tr cid uid]  $\implies$  isPC' (tr@[ $-$ ]) cid uid

```

lemma justify-com:

```

  assumes trace-to tr s
  assumes isPC s cid uid
  shows isPC' tr cid uid
  using assms
proof (induction arbitrary: uid)
  case (empty s) thus ?case
    by (auto simp add: istate-def)

```

next

```

  case (step tr s a ou s')

```

```

  show ?case

```

```

  proof (cases isPC s cid uid)

```

```

    case True with step.IH show ?thesis by (blast intro: isPC'.irrelevant)

```

next

```

  case False note noPC = this

```

```

  show ?thesis proof (cases isChair s' cid uid)

```

```

    case True thus ?thesis

```

```

      by (metis chair justify-chair step.hyps(1) step.hyps(4) trace-to.step)

```

```

next
  case False note noChair=this
  from noPC noChair step.hyps step.prems obtain
    pass uid' where (a=Cact (cPC cid uid' pass uid))
    and isChair s cid uid'
    and [simp]: ou=outOK
    apply (cases a)
    subgoal for x1 apply (cases x1, auto simp add: c-defs) [] .
    subgoal for x2 apply (cases x2, auto simp add: u-defs) [] .
    subgoal for x3 apply (cases x3, auto simp add: uu-defs) [] .
    by simp-all
  thus ?thesis
  apply -
  apply (rule isPC'.add-com, auto simp: isChair-eq[OF <trace-to tr s>]) []
  done
qed
qed
qed

lemma com-justify:
  assumes trace-to tr s
  assumes isPC' tr cid uid
  shows isPC s cid uid
  using assms
proof (induction arbitrary: uid)
  case (empty s) thus ?case
    by (auto simp add: istate-def elim!: isPC'.cases isChair'.cases)
next
  case (step tr s a ou s')
  from step.prems
  show ?case
  proof (cases)
    case chair thus ?thesis
      by (metis isChair-eq isChair-isPC step.hyps(1) step.hyps(3) step.hyps(4))
  trace-to.step)
next
  case (add-com - uid' - - pass)
  hence [simp]: a=Cact (cPC cid uid' pass uid) ou=outOK by simp-all
  from step.hyps show ?thesis
    by (auto simp add: c-defs)
next
  case (irrelevant) with step.IH have COM: isPC s cid uid by simp

from COM <step s a = (ou, s')> show ?thesis
  apply (cases a)
  subgoal for x1 apply (cases x1, auto simp: c-defs) [] .
  subgoal for x2 apply (cases x2, auto simp: u-defs) [] .
  subgoal for x3 apply (cases x3, auto simp: uu-defs) [] .
  by simp-all

```

qed  
qed

**theorem** *isPC-eq*:  $\text{trace-to } tr \ s \implies \text{isPC } s \ cid \ uid = \text{isPC}' \ tr \ cid \ uid$

**using** *justify-com com-justify*  
**by** (*blast*)

## 10.5 Being a Reviewer

**inductive** *isRev'* ::  $(state, act, out) \text{ trans } trace \Rightarrow confID \Rightarrow userID \Rightarrow paperID \Rightarrow \text{bool}$  **where**

*add-rev*:  $\llbracket \text{isChair}' \ tr \ cid \ uid'; \text{trn} = \text{Trans} - (\text{Cact} (\text{cReview } cid \ uid' - pid \ uid)) \text{ outOK} - \rrbracket$

$\implies \text{isRev}' (tr@[trn]) \ cid \ uid \ pid$

| *irrelevant*:  $\llbracket \text{isRev}' \ tr \ cid \ uid \ pid \rrbracket \implies \text{isRev}' (tr@[ - ]) \ cid \ uid \ pid$

**lemma** *justify-rev*:

**assumes** *trace-to tr s*

**assumes** *isRev s cid uid pid*

**shows** *isRev' tr cid uid pid*

**using** *assms*

**proof** (*induction*)

**case** *empty* **thus** *?case*

**by** (*auto simp add: istate-def isRev-def*)

**next**

**case** (*step tr s a ou s'*)

**show** *?case*

**proof** (*cases isRev s cid uid pid*)

**case** *True* **with** *step.IH* **show** *?thesis* **by** (*blast intro: isRev'.irrelevant*)

**next**

**case** *False* **note** *noRev = this*

**with** *step.hyps step.prem* **obtain**

*pass uid'* **where** (*a=Cact (cReview cid uid' pass pid uid)*)

**and** *isChair s cid uid'*

**and** [*simp*]: *ou=outOK*

**apply** (*cases a*)

**subgoal for** *x1* **apply** (*cases x1, auto simp add: c-defs isRev-def*) [] .

**subgoal for** *x2* **apply** (*cases x2, auto simp add: u-defs isRev-def*) [] .

**subgoal for** *x3* **apply** (*cases x3, auto simp add: uu-defs isRev-def*) [] .

**by** *simp-all*

**thus** *?thesis*

**apply** -

**apply** (*rule isRev'.add-rev, auto simp: isChair-eq[OF <trace-to tr s>]*) []

**done**

qed  
qed

```

lemma rev-justify:
  assumes trace-to tr s
  assumes isRev' tr cid uid pid
  shows isRev s cid uid pid
  using assms
proof (induction arbitrary: uid)
  case (empty s) thus ?case
    by (auto simp add: istate-def elim!: isRev'.cases)
next
  case (step tr s a ou s')
  from step.prems
  show ?case
  proof (cases)
    case (add-rev - uid' - - pass)
    hence [simp]: a=Cact (cReview cid uid' pass pid uid) ou=outOK by simp-all
    from step.hyps show ?thesis
      by (auto simp add: c-defs isRev-def)
  next
  case (irrelevant) with step.IH have REV: isRev s cid uid pid by simp

  note roles-confIDs[OF step.hyps(2)]
  with REV ⟨step s a = (ou, s')⟩ show ?thesis
    apply (cases a)
    subgoal for x1 apply (cases x1, auto simp: c-defs isRev-def) [] .
    subgoal for x2 apply (cases x2, auto simp: u-defs isRev-def) [] .
    subgoal for x3 apply (cases x3, auto simp: uu-defs isRev-def) [] .
    by simp-all
  qed
qed

theorem isRev-eq: trace-to tr s ⟹ isRev s cid uid pid = isRev' tr cid uid pid

  using justify-rev rev-justify
  by (blast)

```

## 10.6 Conflicts

```

fun irrev-conflict :: userID ⇒ paperID ⇒ (state,act,out) trans ⇒ bool

```

**where**

```

  irrev-conflict uid pid (Trans - (Cact (cPaper - uid' - pid' - -)) outOK -)
    ⟷ uid'=uid ∧ pid'=pid
| irrev-conflict uid pid (Trans - (Cact (cAuthor - - - pid' uid')) outOK -)
    ⟷ uid'=uid ∧ pid'=pid
| irrev-conflict uid pid - ⟷ False

```

```

fun set-conflict :: userID ⇒ paperID ⇒ (state,act,out) trans ⇒ bool

```

**where**

```

    set-conflict uid pid (Trans - (Cact (cConflict - - - pid' uid')) outOK -)
       $\longleftrightarrow$  uid'=uid  $\wedge$  pid'=pid
  | set-conflict uid pid (Trans - (Uact (uPref - uid' - pid' Conflict)) outOK -)
       $\longleftrightarrow$  uid'=uid  $\wedge$  pid'=pid
  | set-conflict - - -  $\longleftrightarrow$  False

```

**fun** reset-conflict :: userID  $\Rightarrow$  paperID  $\Rightarrow$  (state,act,out)trans  $\Rightarrow$  bool

**where**

```

    reset-conflict uid pid (Trans - (Uact (uPref - uid' - pid' pr)) outOK -)
       $\longleftrightarrow$  uid'=uid  $\wedge$  pid'=pid  $\wedge$  pr $\neq$ Conflict
  | reset-conflict - - -  $\longleftrightarrow$  False

```

**definition** conflict-trace :: userID  $\Rightarrow$  paperID  $\Rightarrow$  (state,act,out) trans trace  $\Rightarrow$  bool

**where**

```

    conflict-trace uid pid tr  $\equiv$ 
      ( $\exists$  trn $\in$ set tr. irrev-conflict uid pid trn)
 $\vee$  ( $\exists$  tr1 trn tr2. tr=tr1@trn#tr2  $\wedge$ 
      set-conflict uid pid trn  $\wedge$  ( $\forall$  trn $\in$ set tr2.  $\neg$ reset-conflict uid pid trn))

```

**lemma** irrev-conflict-impl-author:

**assumes** trace-to tr s

**assumes**  $\exists$  trn $\in$ set tr. irrev-conflict uid pid trn

**shows**  $\exists$  cid. isAut s cid uid pid

**using** assms

**apply** induction

**apply** (auto simp add: istate-def)  $\square$

**subgoal for** - - a **apply** (cases a)

**subgoal for** x1 **apply** (cases x1, auto simp: c-defs, (metis roles-confIDs)+)  $\square$

**subgoal for** x2 **apply** (cases x2, auto simp: u-defs)  $\square$  .

**subgoal for** x3 **apply** (cases x3, auto simp: uu-defs)  $\square$  .

**by** simp-all

**done**

**lemma** irrev-conflict-impl-conflict:

**assumes** trace-to tr s

**assumes**  $\exists$  trn $\in$ set tr. irrev-conflict uid pid trn

**shows** pref s uid pid = Conflict

**by** (metis assms(1) assms(2) irrev-conflict-impl-author

isAut-pref-Conflict reach-trace-to-conv)

**lemma** conflict-justify:

**assumes** TR: trace-to tr s

**assumes** conflict-trace uid pid tr

**shows** pref s uid pid = Conflict

**using** assms(2)

**unfolding** conflict-trace-def

```

proof (cases rule: disjE[consumes 1, case-names irrev set])
  case irrev thus ?thesis by (simp add: irrev-conflict-impl-conflict[OF TR])
next
  case set
  then obtain tr1 trn tr2 where
    [simp]: tr = tr1 @ trn # tr2 and
    SET: set-conflict uid pid trn
    and NRESET:  $\forall trn \in set\ tr2. \neg reset-conflict\ uid\ pid\ trn$ 
  by blast

from TR obtain s1 s2 a ou where
  [simp]: trn = Trans s1 a ou s2 and
  TR1: trace-to tr1 s1 and
  STEP: step s1 a = (ou,s2) and
  TR2: trace-between s2 tr2 s
  by (fastforce simp add: trace-to-ft trace-ft-simps)

from STEP SET have pref s2 uid pid = Conflict
  apply (cases a)
  subgoal for x1 apply (cases x1, auto simp: c-defs) [] .
  subgoal for x2 apply (cases x2, auto simp: u-defs) [] .
  subgoal for - x65 apply (cases x65, auto) [] .
  subgoal for - - x65 apply (cases x65, auto) [] .
  subgoal for - - - x65 apply (cases x65, auto) [] . .
  by simp-all

with TR2 NRESET show ?thesis
  apply induction
  subgoal by simp
  subgoal for - - - a apply (cases a)
    subgoal for x1 apply (cases x1, auto simp: c-defs) [] .
    subgoal for x2 apply (cases x2, auto simp: u-defs) [] .
    subgoal for x3 apply (cases x3, auto simp: uu-defs) [] .
  by simp-all
  done
qed

lemma justify-conflict:
  assumes TR: trace-to tr s
  assumes pref s uid pid = Conflict
  shows conflict-trace uid pid tr
  using assms
proof induction
  case empty thus ?case by (auto simp add: istate-def)
next
  case (step tr s a ou s')

  let ?trn = Trans s a ou s'

```

```

show ?case proof (cases pref s uid pid = Conflict)
  case False
  with step.premis ⟨step s a = (ou, s')⟩
  have irrev-conflict uid pid ?trn ∨ set-conflict uid pid ?trn
  apply (cases a)
  subgoal for x1 apply (cases x1, auto simp: c-defs) [] .
  subgoal for x2 apply (cases x2, auto simp: u-defs) [] .
  subgoal for x3 apply (cases x3, auto simp: uu-defs) [] .
  by simp-all
  thus ?thesis
  unfolding conflict-trace-def by fastforce
next
  case True with step.IH have CT: conflict-trace uid pid tr .

  from step.premis ⟨step s a = (ou, s')⟩ have ¬reset-conflict uid pid ?trn
  apply (cases a)
  subgoal by simp
  subgoal for x2 by (cases x2, auto simp: u-defs)
  by simp-all

  thus ?thesis using CT
  unfolding conflict-trace-def
  apply clarsimp
  by (metis rotate1.simps(2) set-ConsD set-rotate1)

qed
qed

```

```

theorem conflict-eq:
  assumes trace-to tr s
  shows pref s uid pid = Conflict  $\longleftrightarrow$  conflict-trace uid pid tr
  using assms conflict-justify justify-conflict by auto

```

## 10.7 Conference Phases

**fun** is-uPhase **where**

```

  is-uPhase cid (Trans - (Uact (uConfA cid' -) outOK -))  $\longleftrightarrow$  cid'=cid
| is-uPhase cid (Trans - (Uact (uPhase cid' - -) outOK -))  $\longleftrightarrow$  cid'=cid
| is-uPhase - -  $\longleftrightarrow$  False

```

**inductive** phase' :: (state,act,out) trans trace  $\Rightarrow$  confID  $\Rightarrow$  nat  $\Rightarrow$  bool **where**

```

  initial: phase' [] cid noPH
| approve: [[phase' tr cid noPH; trn=Trans s (Uact (uConfA cid (voronkov s) -))
outOK - ]
 $\implies$  phase' (tr@[trn]) cid setPH
| advance: [trn = (Trans - (Uact (uPhase cid uid - ph)) outOK -); isChair' tr cid
uid]
 $\implies$  phase' (tr@[trn]) cid ph
| irrelevant: [phase' tr cid ph; ¬is-uPhase cid trn ]  $\implies$  phase' (tr@[trn]) cid ph

```

```

lemma justify-phase:
  assumes trace-to tr s
  assumes phase s cid = ph
  shows phase' tr cid ph
  using assms
proof (induction arbitrary: ph)
  case (empty s) thus ?case
    by (auto simp add: istate-def phase'.initial)
next
  case (step tr s a ou s')
  thus ?case
    apply (cases a)
    subgoal for x1 apply (cases x1, auto simp: c-defs intro: phase'.advance
phase'.irrelevant) [] .
    subgoal for x2 apply (cases x2,
      auto
      simp: u-defs isChair-eq
      intro: phase'.advance phase'.irrelevant phase'.approve,
      (fastforce intro: phase'.approve phase'.irrelevant phase'.advance)+
    ) [] .

    subgoal for x3 apply (cases x3, auto simp: uu-defs intro: phase'.irrelevant) []
    .
  by (auto intro: phase'.advance phase'.irrelevant)
qed

lemma phase-justify:
  assumes trace-to tr s
  assumes phase' tr cid ph
  shows phase s cid = ph
  using assms
proof (induction arbitrary: ph)
  case (empty s) thus ?case
    by (auto simp add: istate-def elim: phase'.cases)
next
  case (step tr s a ou s')
  from step.prem1
  show ?case
  proof (cases)
    case (approve - - - pass -)
    hence [simp]: a=Uact (uConfA cid (voronkov s) pass) ou=outOK ph=setPH
  by simp-all
  from step.hyps show ?thesis
    by (auto simp add: u-defs)
  next
  case (advance - - uid pass -)
  hence [simp]: a=Uact (uPhase cid uid pass ph) ou=outOK
  from step.hyps show ?thesis

```

```

    by (auto simp add: u-defs)
  next
    case (irrelevant) with step.IH have PH: phase s cid = ph  $\neg$  is-uPhase cid
    (Trans s a ou s')
    by simp-all

    from PH ⟨step s a = (ou, s')⟩ show ?thesis
    apply (cases a)
    subgoal for x1 apply (cases x1, auto simp: c-defs) [] .
    subgoal for x2 apply (cases x2, auto simp: u-defs) [] .
    subgoal for x3 apply (cases x3, auto simp: uu-defs) [] .
    by simp-all
  qed auto
qed

```

```

theorem phase-eq:
  assumes trace-to tr s
  shows phase s cid = ph  $\longleftrightarrow$  phase' tr cid ph
  using assms phase-justify justify-phase by blast

```

```

end
theory All-BD-Security-Instances-for-CoCon
imports

```

*Paper-Confidentiality/Paper-All*

*Review-Confidentiality/Review-All*

*Discussion-Confidentiality/Discussion-All*

*Decision-Confidentiality/Decision-All*

*Reviewer-Assignment-Confidentiality/Reviewer-Assignment-All*

*Traceback-Properties*

```
begin
```

```
end
```

## References

- [1] T. Bauereiß, A. Pesenti Gritti, A. Popescu, and F. Raimondi. Cosmed: A confidentiality-verified social media platform. In J. C. Blanchette and

- S. Merz, editors, *Interactive Theorem Proving - 7th International Conference, ITP 2016, Nancy, France, August 22-25, 2016, Proceedings*, volume 9807 of *Lecture Notes in Computer Science*, pages 87–106. Springer, 2016.
- [2] T. Bauereiß, A. Pesenti Gritti, A. Popescu, and F. Raimondi. Cosmed: A confidentiality-verified social media platform. *J. Autom. Reason.*, 61(1-4):113–139, 2018.
- [3] S. Kanav, P. Lammich, and A. Popescu. A conference management system with verified document confidentiality. In A. Biere and R. Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 2014.
- [4] A. Popescu, T. Bauereiss, and P. Lammich. Bounded-Deducibility security (invited paper). In L. Cohen and C. Kaliszyk, editors, *12th International Conference on Interactive Theorem Proving, ITP 2021, June 29 to July 1, 2021, Rome, Italy (Virtual Conference)*, volume 193 of *LIPICs*, pages 3:1–3:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- [5] A. Popescu, P. Lammich, and T. Bauereiss. Bounded-deducibility security. In G. Klein, T. Nipkow, and L. Paulson, editors, *Archive of Formal Proofs*, 2014.
- [6] A. Popescu, P. Lammich, and P. Hou. Cocon: A conference management system with formally verified document confidentiality. *J. Autom. Reason.*, 65(2):321–356, 2021.