

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

1	Introduction	3
1.1	The basic types	5
1.2	Conference, user and paper IDs	7
2	System specification	9
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)	12
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	21
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	26
3	Safety properties	41
3.1	Infrastructure for invariance reasoning	41
3.2	Safety proofs	43
3.3	Properties of the step function	57
3.4	Action-safety properties	58
3.5	Miscellaneous	58
4	Observation setup for confidentiality properties	61
5	Paper Confidentiality	62
5.1	Preliminaries	62
5.2	Value Setup	66
5.3	Confidentiality protection from users who are not the paper's authors or PC members	67
5.4	Confidentiality protection from non-authors	70
6	Review Confidentiality	72
6.1	Preliminaries	73
6.2	Value Setup	81
6.3	Confidentiality protection from users who are not the review's author	83
6.4	Confidentiality protection from users who are not the review's author or a PC member	86
6.5	Confidentiality from users who are not the review's author, a PC member, or an author of the paper	90
7	Discussion Confidentiality	93
7.1	Preliminaries	94
7.2	Value Setup	97
7.3	Confidentiality protection from non-PC-members	98
8	Decision Confidentiality	101
8.1	Preliminaries	101
8.2	Value Setup	108
8.3	Confidentiality protection from non-PC-members	110
8.4	Confidentiality protection from users who are not PC mem- bers or authors of the paper	113
9	Reviewer Assignment Confidentiality	116
9.1	Preliminaries	117
9.2	Value Setup	126
9.3	Confidentiality protection from non-PC-members	128

9.4 Confidentiality protection from users who are not PC members or authors of the paper	133
10 Traceback properties	138
10.1 Preliminaries	138
10.2 Authorship	139
10.3 Becoming a Conference Chair	140
10.4 Committee Membership	141
10.5 Being a Reviewer	141
10.6 Conflicts	142
10.7 Conference Phases	143

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,¹ 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

¹The 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*,² 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 “ σ ”) 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*

²In 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
  <proof>

lemma member-userIDAsStr-iff[simp]: str  $\in$  userIDAsStr ' (set userIDs)  $\longleftrightarrow$ 
  UserID str  $\in\in$  userIDs
  <proof>

lemma getFreshUserID:  $\neg$  getFreshUserID userIDs  $\in\in$  userIDs
  <proof>

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 <proof>
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
  <proof>

lemma member-paperIDAsStr-iff[simp]: str  $\in$  paperIDAsStr ' paperIDs  $\longleftrightarrow$  Pa-
  perID str  $\in$  paperIDs
  <proof>

lemma getFreshPaperID:  $\neg$  getFreshPaperID paperIDs  $\in\in$  paperIDs
  <proof>

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 <proof>
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*
<proof>

lemma *member-confIDAsStr-iff[simp]*: *str* \in *confIDAsStr* ‘(*set confIDs*) \longleftrightarrow *ConfID str* \in *confIDs*
<proof>

lemma *getFreshConfID*: \neg *getFreshConfID confIDs* \in *confIDs*
<proof>

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 *<proof>*

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* \Rightarrow *password*

user :: *userID* \Rightarrow *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 uID* ≡ *PC* ∈∈ *roles s confID uID*

abbreviation *isChair* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *bool* **where**

isChair *s confID uID* ≡ *Chair* ∈∈ *roles s confID uID*

abbreviation *isAut* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *paperID* ⇒ *bool* **where**

isAut *s confID uID papID* ≡ *Aut papID* ∈∈ *roles s confID uID*

definition *isAutSome* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *bool* **where**

isAutSome *s confID uID* ≡ *list-ex* (*isAut* *s confID uID*) (*paperIDs* *s confID*)

definition *authors* :: *state* ⇒ *confID* ⇒ *paperID* ⇒ *userID list* **where**

authors *s confID papID* ≡ *filter* (λ *uID*. *isAut* *s confID uID papID*) (*userIDs* *s*)

abbreviation *isRevNth* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *paperID* ⇒ *nat* ⇒ *bool*

where

isRevNth *s confID uID papID n* ≡ *Rev papID n* ∈∈ *roles s confID uID*

definition *isRev* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *paperID* ⇒ *bool* **where**

isRev *s confID uID papID* ≡ *list-ex* (*isRevRoleFor* *papID*) (*roles s confID uID*)

definition *getRevRole* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *paperID* ⇒ *role option*

where

getRevRole *s confID uID papID* ≡ *List.find* (*isRevRoleFor* *papID*) (*roles s confID uID*)

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* ≡

case *getRevRole* *s confID uID papID* of *Some* (*Rev papID'* *n*) ⇒ *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*
<proof>

lemma *getRevRole-Some*:
assumes *getRevRole* *s* *confID* *uID* *papID* = *Some* *role*
shows \exists *n*. *role* = *Rev* *papID* *n*
<proof>

lemma *isRev-getRevRole2*:
assumes *isRev* *s* *confID* *uID* *papID***shows** \exists *n*. *getRevRole* *s* *confID* *uID* *papID* =
Some (*Rev* *papID* *n*)
<proof>

lemma *isRev-imp-isRevNth-getReviewIndex*:
assumes *isRev* *s* *confID* *uID* *papID*
shows *isRevNth* *s* *confID* *uID* *papID* (*getReviewIndex* *s* *confID* *uID* *papID*)
<proof>

lemma *isRev-def2*:
isRev *s* *confID* *uID* *papID* \longleftrightarrow (\exists *n*. *isRevNth* *s* *confID* *uID* *papID* *n*) (**is** ?*A* \longleftrightarrow
 ?*B*)
<proof>

lemma *isRev-def3*:
isRev *s* *confID* *uID* *papID* \longleftrightarrow *isRevNth* *s* *confID* *uID* *papID* (*getReviewIndex* *s*
confID *uID* *papID*)
<proof>

lemma *getFreshPaperID-getAllPaperIDs[simp]*:
assumes *confID* \in *confIDs* *s*

shows $\neg \text{getFreshPaperID } (\text{getAllPaperIDs } s) \in \in \text{paperIDs } s \text{ confID}$
 $\langle \text{proof} \rangle$

lemma *getRevRole-Some-Rev*:

$\text{getRevRole } s \text{ cid uid pid} = \text{Some } (\text{Rev } \text{pid}' \ n) \implies \text{pid}' = \text{pid}$
 $\langle \text{proof} \rangle$

lemma *getRevRole-Some-Rev-isRevNth*:

$\text{getRevRole } s \text{ cid uid pid} = \text{Some } (\text{Rev } \text{pid}' \ n) \implies \text{isRevNth } s \text{ cid uid pid } n$
 $\langle \text{proof} \rangle$

definition *IDsOK* :: $\text{state} \Rightarrow \text{confID list} \Rightarrow \text{userID list} \Rightarrow \text{paperID list} \Rightarrow \text{bool}$
where

$\text{IDsOK } s \ \text{cIDs } \ \text{uIDs } \ \text{papIDs} \equiv$
 $\text{list-all } (\lambda \ \text{confID}. \ \text{confID} \in \in \text{confIDs } s) \ \text{cIDs} \ \wedge$
 $\text{list-all } (\lambda \ \text{uID}. \ \text{uID} \in \in \text{userIDs } s) \ \text{uIDs} \ \wedge$
 $\text{list-all } (\lambda \ \text{papID}. \ \text{papID} \in \in \text{paperIDs } s \ (\text{hd } \text{cIDs})) \ \text{papIDs}$

2.2 The actions

2.2.1 Initialization of the system

definition *istate* :: state

where

$\text{istate} \equiv$
 \langle
 $\text{confIDs} = [],$
 $\text{conf} = (\lambda \ \text{confID}. \ \text{emptyConf}),$
 $\text{userIDs} = [\text{voronkovUserID}],$
 $\text{pass} = (\lambda \ \text{uID}. \ \text{emptyPass}),$
 $\text{user} = (\lambda \ \text{uID}. \ \text{emptyUser}),$
 $\text{roles} = (\lambda \ \text{confID } \ \text{uID}. \ []),$
 $\text{paperIDs} = (\lambda \ \text{confID}. \ []),$
 $\text{paper} = (\lambda \ \text{papID}. \ \text{emptyPaper}),$
 $\text{pref} = (\lambda \ \text{uID } \ \text{papID}. \ \text{NoPref}),$
 $\text{voronkov} = \text{voronkovUserID},$
 $\text{news} = (\lambda \ \text{confID}. \ []),$
 $\text{phase} = (\lambda \ \text{confID}. \ \text{noPH})$
 \rangle

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

definition *createUser* :: $\text{state} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{string} \Rightarrow \text{string} \Rightarrow \text{string}$
 $\Rightarrow \text{state}$

where

$\text{createUser } s \ \text{uID } \ p \ \text{name } \ \text{info } \ \text{email} \equiv$
 $\text{let } \ \text{uIDs} = \text{userIDs } s$
 in
 $s \ (\text{userIDs} := \text{uID } \# \ \text{uIDs},$

$user := (user\ s)\ (uID := User\ name\ info\ email),$
 $pass := (pass\ s)\ (uID := p))$

definition $e\text{-createUser} :: state \Rightarrow userID \Rightarrow password \Rightarrow string \Rightarrow string \Rightarrow string \Rightarrow bool$ **where**
 $e\text{-createUser}\ s\ uID\ p\ name\ info\ email \equiv$
 $\neg\ uID \in\in\ userIDs\ s$

definition $updateUser :: state \Rightarrow userID \Rightarrow password \Rightarrow password \Rightarrow string \Rightarrow string \Rightarrow string \Rightarrow state$
where
 $updateUser\ s\ uID\ p\ p'\ name\ info\ email \equiv$
 $s\ (user := (user\ s)\ (uID := User\ name\ info\ email),$
 $pass := (pass\ s)\ (uID := p'))$

definition $e\text{-updateUser} :: state \Rightarrow userID \Rightarrow password \Rightarrow password \Rightarrow string \Rightarrow string \Rightarrow bool$
where
 $e\text{-updateUser}\ s\ uID\ p\ p'\ name\ info\ email \equiv$
 $IDsOK\ s\ [uID] \wedge\ pass\ s\ uID = p$

definition $readAmIVoronkov :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
 $readAmIVoronkov\ s\ uID\ p \equiv$
 $uID = voronkov\ s$

definition $e\text{-readAmIVoronkov} :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
 $e\text{-readAmIVoronkov}\ s\ uID\ p \equiv$
 $IDsOK\ s\ [uID] \wedge\ pass\ s\ uID = p$

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 $e\text{-readUser} :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow bool$
where
 $e\text{-readUser}\ s\ uID\ p\ uID' \equiv$
 $IDsOK\ s\ [uID, uID'] \wedge\ pass\ s\ uID = p$

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

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

2.2.3 Actions available in the noPH phase

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

where

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

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

where

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

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
 $s \ (\!| \ roles := fun\text{-upd}2 \ (roles \ s) \ confID \ uID' \ (List.insert \ PC \ rls) \ |)$

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

where

$e\text{-createPC} \ s \ confID \ uID \ p \ uID' \equiv$
 $let \ uIDs = userIDs \ s$
 in
 $IDsOK \ s \ [confID] \ [uID, uID'] \ [] \ \wedge \ pass \ s \ uID = p \ \wedge$
 $phase \ s \ confID = setPH \ \wedge \ isChair \ s \ confID \ uID$

definition $createChair :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow state$

where

$createChair \ s \ confID \ uID \ p \ uID' \equiv$
 $let \ rls = roles \ s \ confID \ uID'$
 in
 $s \ (\!| \ roles := fun\text{-upd}2 \ (roles \ s) \ confID \ uID' \ (List.insert \ PC \ (List.insert \ Chair \ rls)) \ |)$

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

where

$e\text{-createChair} \ s \ confID \ uID \ p \ uID' \equiv$
 $let \ uIDs = userIDs \ s$
 in
 $IDsOK \ s \ [confID] \ [uID, uID'] \ [] \ \wedge \ pass \ s \ uID = p \ \wedge$

$phase\ s\ confID = setPH \wedge isChair\ s\ confID\ uID$

2.2.5 Actions available starting from the setPH phase

definition $updatePhase :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow phase \Rightarrow state$
where

$updatePhase\ s\ confID\ uID\ p\ ph \equiv$
 $s\ (\backslash phase := (phase\ s)\ (confID := ph))$

definition $e-updatePhase :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow phase \Rightarrow bool$
where

$e-updatePhase\ s\ confID\ uID\ p\ ph \equiv$
 $IDsOK\ s\ [confID]\ [uID]\ [] \wedge pass\ s\ uID = p \wedge$
 $phase\ s\ confID \geq setPH \wedge phase\ s\ confID < closedPH \wedge isChair\ s\ confID\ uID \wedge$
 $ph = SucPH\ (phase\ s\ confID)$

definition $uupdateNews :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow string \Rightarrow state$

where

$uupdateNews\ s\ confID\ uID\ p\ ev \equiv$
 $let\ evs = news\ s\ confID$
 in
 $s\ (\backslash news := (news\ s)\ (confID := ev \# evs))$

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

where

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

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

where

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

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

where

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

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 $e-listPC :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow bool$

where

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

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 $e-listChair :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow bool$

where

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

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

$createPaper\ s\ confID\ uID\ p\ papID\ title\ abstract \equiv$
 $let\ papIDs = paperIDs\ s\ confID;$
 $rls = roles\ s\ confID\ uID$
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 $e-createPaper :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow string \Rightarrow string \Rightarrow bool$

where

$e-createPaper\ s\ confID\ uID\ p\ papID\ name\ info \equiv$
 $IDsOK\ s\ [confID]\ [uID]\ [] \wedge pass\ s\ uID = p \wedge$
 $phase\ s\ confID = subPH \wedge$
 $\neg papID \in \in\ getAllPaperIDs\ s$

definition $createAuthor :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow userID \Rightarrow state$

where

$createAuthor\ s\ confID\ uID\ p\ papID\ uID' \equiv$
 $let\ rls = roles\ s\ confID\ uID'$

in
 s ($\text{roles} := \text{fun-upd2}$ ($\text{roles } s$) $\text{confID } uID'$ (List.insert ($\text{Aut } \text{papID}$) rls),
 $\text{pref} := \text{fun-upd2}$ ($\text{pref } s$) 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

$e\text{-createAuthor } s \text{ confID } uID \text{ p papID } uID' \equiv$
 $IDsOK \ s \ [confID] \ [uID, uID'] \ [papID] \wedge \text{pass } s \ uID = p \wedge$
 $\text{phase } s \ \text{confID} = \text{subPH} \wedge \text{isAut } s \ \text{confID } uID \ \text{papID} \wedge uID \neq uID'$

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

$\text{updatePaperTA } s \ \text{confID } uID \ \text{p } \text{papID} \ \text{title} \ \text{abstract} \equiv$
 $\text{case } \text{paper } s \ \text{papID} \ \text{of } \text{Paper } \text{title}' \ \text{abstract}' \ \text{pc} \ \text{reviews} \ \text{dis} \ \text{decs} \Rightarrow$
 $s \ (\text{paper} := (\text{paper } s) \ (\text{papID} := \text{Paper } \text{title} \ \text{abstract} \ \text{pc} \ \text{reviews} \ \text{dis} \ \text{decs}))$

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

$e\text{-updatePaperTA } s \ \text{confID } uID \ \text{p } \text{papID} \ \text{name} \ \text{info} \equiv$
 $IDsOK \ s \ [confID] \ [uID] \ [papID] \wedge \text{pass } s \ uID = p \wedge$
 $\text{phase } s \ \text{confID} = \text{subPH} \wedge \text{isAut } s \ \text{confID } uID \ \text{papID}$

definition $\text{updatePaperC} :: \text{state} \Rightarrow \text{confID} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{paperID} \Rightarrow \text{pcontent} \Rightarrow \text{state}$

where

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

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

where

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

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

where

$\text{createConflict } s \ \text{confID } uID \ \text{p } \text{papID } uID' \equiv$
 $s \ (\text{pref} := \text{fun-upd2} \ (\text{pref } s) \ uID' \ \text{papID} \ \text{Conflict})$

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

where

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

2.2.7 Actions available starting from the subPH phase

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

where

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

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

where

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

definition $\text{readPaperC} :: \text{state} \Rightarrow \text{confID} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{paperID} \Rightarrow \text{pcontent}$

where

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

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

where

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

definition $\text{listPapers} :: \text{state} \Rightarrow \text{confID} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{paperID list}$

where

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

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

where

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

definition $\text{listMyPapers} :: \text{state} \Rightarrow \text{confID} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{paperID list}$

where

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

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

where

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

2.2.8 Actions available in the bidPH phase

definition $\text{updatePref} :: \text{state} \Rightarrow \text{confID} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{paperID} \Rightarrow$
 $\text{preference} \Rightarrow \text{state}$

where

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

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

where

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

2.2.9 Actions available starting from the bidPH phase

definition $\text{readPref} :: \text{state} \Rightarrow \text{confID} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{paperID} \Rightarrow \text{pref-}$
 erence

where

$\text{readPref } s \ \text{confID } uID \text{ p } \text{papID} \equiv$
 $\text{pref } s \ uID \ \text{papID}$

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

where

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

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

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

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

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

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

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

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

where

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

definition $uupdateReview ::$

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

where

$uupdateReview \ s \ confID \ uID \ p \ papID \ n \ rc \equiv$
 $case \ paper \ s \ papID \ of \ Paper \ title \ abstract \ pc \ reviews \ dis \ decs \Rightarrow$
 $let \ review = rc \ \# \ (reviews \ ! \ n); \ reviews' = list\text{-update} \ reviews \ n \ review$
 in
 $s \ (\backslash paper := fun\text{-upd} \ (paper \ s) \ papID \ (Paper \ title \ abstract \ pc \ reviews' \ dis \ decs))$

definition $e\text{-uupdateReview} ::$

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

where

$e\text{-uupdateReview} \ s \ confID \ uID \ p \ papID \ n \ rc \equiv$
 $IDsOK \ s \ [confID] \ [uID] \ [papID] \ \wedge \ pass \ s \ uID = p \ \wedge$
 $phase \ s \ confID = disPH \ \wedge \ isRev \ s \ confID \ uID \ papID \ \wedge$
 $getReviewIndex \ s \ confID \ uID \ papID = n$

definition $uupdateDec :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow$
 $decision \Rightarrow state$

where

$uupdateDec \ s \ confID \ uID \ p \ papID \ dec \equiv$
 $case \ paper \ s \ papID \ of \ Paper \ title \ abstract \ pc \ reviews \ dis \ decs \Rightarrow$
 $s \ (\backslash paper := fun\text{-upd} \ (paper \ s) \ papID \ (Paper \ title \ abstract \ pc \ reviews \ dis \ (dec \ \#$
 $decs)))$

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

where

$e\text{-updateDec } s \text{ confID } uID \text{ } p \text{ } papID \text{ } dec \equiv$
 $IDsOK \ s \ [confID] \ [uID] \ [papID] \wedge \text{pass } s \ uID = p \wedge$
 $\text{phase } s \ \text{confID} = \text{disPH} \wedge \text{isChair } s \ \text{confID } uID \wedge \text{pref } s \ uID \ \text{papID} \neq \text{Conflict}$

2.2.13 Actions available starting from the disPH phase

definition $\text{readReviews} :: \text{state} \Rightarrow \text{confID} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{paperID} \Rightarrow (\text{userID} * \text{review}) \text{ list}$

where

$\text{readReviews } s \ \text{confID } uID \text{ } p \text{ } papID \equiv$
 $\text{getReviewersReviews } s \ \text{confID } papID$

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

where

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

definition $\text{readDecs} :: \text{state} \Rightarrow \text{confID} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{paperID} \Rightarrow \text{decision list}$

where

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

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

where

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

definition $\text{readDis} :: \text{state} \Rightarrow \text{confID} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{paperID} \Rightarrow \text{string list}$

where

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

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

where

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

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

$readFinalDec\ s\ confID\ uID\ p\ papID \equiv$
 $case\ paper\ s\ papID\ of\ Paper\ title\ abstract\ pc\ reviews\ dis\ decs \Rightarrow$
 $case\ decs\ of\ [] \Rightarrow NoDecision\ | dec \# decs \Rightarrow dec$

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

where

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

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 userID p name info email) =
  (if e-createUser s userID p name info email
   then (outOK, createUser s userID 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 userID p p' name info email) =
  (if e-updateUser s userID p p' name info email
   then (outOK, updateUser s userID 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 ⇒ rAct ⇒ 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  $\Rightarrow$  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)  
   $\langle$ proof $\rangle$ 
```

```
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  
  
 $\langle$ proof $\rangle$ 
```

```
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
(*proof*)

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)
(*proof*)

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

assumes

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

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

and

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

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

and

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

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

and

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

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

and

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

$$\begin{aligned} & \llbracket \text{reach } s; \varphi \ s; \text{e-createPaper } s \ \text{confID } uID \ p \ \text{papID } \text{name } \text{info} \rrbracket \\ & \implies \varphi \ (\text{createPaper } s \ \text{confID } uID \ p \ \text{papID } \text{name } \text{info}) \end{aligned}$$

and

$$\begin{aligned} & \bigwedge s \ \text{confID } uID \ p \ \text{papID } uID'. \\ & \llbracket \text{reach } s; \varphi \ s; \text{e-createAuthor } s \ \text{confID } uID \ p \ \text{papID } uID \rrbracket \\ & \implies \varphi \ (\text{createAuthor } s \ \text{confID } uID \ p \ \text{papID } uID') \end{aligned}$$

and

$$\begin{aligned} & \bigwedge s \ \text{confID } uID \ p \ \text{papID } uID'. \\ & \llbracket \text{reach } s; \varphi \ s; \text{e-createConflict } s \ \text{confID } uID \ p \ \text{papID } uID \rrbracket \\ & \implies \varphi \ (\text{createConflict } s \ \text{confID } uID \ p \ \text{papID } uID') \end{aligned}$$

and

$$\begin{aligned} & \bigwedge s \ \text{confID } uID \ p \ \text{papID } uID'. \\ & \llbracket \text{reach } s; \varphi \ s; \text{e-createReview } s \ \text{confID } uID \ p \ \text{papID } uID \rrbracket \\ & \implies \varphi \ (\text{createReview } s \ \text{confID } uID \ p \ \text{papID } uID') \end{aligned}$$

shows $cIsInvar \ \varphi$

<proof>

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

assumes

$$\begin{aligned} & \bigwedge s \ uID \ p \ p' \ \text{name } \text{info } \text{email}. \\ & \llbracket \text{reach } s; \varphi \ s; \text{e-updateUser } s \ uID \ p \ p' \ \text{name } \text{info } \text{email} \rrbracket \\ & \implies \varphi \ (\text{updateUser } s \ uID \ p \ p' \ \text{name } \text{info } \text{email}) \end{aligned}$$

and

$$\begin{aligned} & \bigwedge s \ \text{confID } uID \ p. \\ & \llbracket \text{reach } s; \varphi \ s; \text{e-updateConfA } s \ \text{confID } uID \ p \rrbracket \implies \varphi \ (\text{updateConfA } s \ \text{confID } \\ & \text{uID } p) \end{aligned}$$

and

$$\begin{aligned} & \bigwedge s \ \text{confID } uID \ p \ ph. \\ & \llbracket \text{reach } s; \varphi \ s; \text{e-updatePhase } s \ \text{confID } uID \ p \ ph \rrbracket \implies \varphi \ (\text{updatePhase } s \ \text{confID } \\ & \text{uID } p \ ph) \end{aligned}$$

and

$$\begin{aligned} & \bigwedge s \ \text{confID } uID \ p \ \text{paperID } \text{name } \text{info}. \\ & \llbracket \text{reach } s; \varphi \ s; \text{e-updatePaperTA } s \ \text{confID } uID \ p \ \text{paperID } \text{name } \text{info} \rrbracket \\ & \implies \varphi \ (\text{updatePaperTA } s \ \text{confID } uID \ p \ \text{paperID } \text{name } \text{info}) \end{aligned}$$

and

$$\begin{aligned} & \bigwedge s \ \text{confID } uID \ p \ \text{paperID } pc. \\ & \llbracket \text{reach } s; \varphi \ s; \text{e-updatePaperC } s \ \text{confID } uID \ p \ \text{paperID } pc \rrbracket \\ & \implies \varphi \ (\text{updatePaperC } s \ \text{confID } uID \ p \ \text{paperID } pc) \end{aligned}$$

and

$$\begin{aligned} & \bigwedge s \ \text{confID } uID \ p \ \text{paperID } \text{preference}. \\ & \llbracket \text{reach } s; \varphi \ s; \text{e-updatePref } s \ \text{confID } uID \ p \ \text{paperID } \text{preference} \rrbracket \\ & \implies \varphi \ (\text{updatePref } s \ \text{confID } uID \ p \ \text{paperID } \text{preference}) \end{aligned}$$

and

$$\begin{aligned} & \bigwedge s \ \text{confID } uID \ p \ \text{paperID } n \ rc. \\ & \llbracket \text{reach } s; \varphi \ s; \text{e-updateReview } s \ \text{confID } uID \ p \ \text{paperID } n \ rc \rrbracket \\ & \implies \varphi \ (\text{updateReview } s \ \text{confID } uID \ p \ \text{paperID } n \ rc) \end{aligned}$$

and

$$\bigwedge s \ \text{confID } uID \ p \ \text{paperID } fpc.$$

$$\begin{aligned} & \llbracket \text{reach } s; \varphi \ s; \text{e-updateFPaperC } s \ \text{confID } uID \ p \ \text{paperID } fpc \rrbracket \\ & \implies \varphi \ (\text{updateFPaperC } s \ \text{confID } uID \ p \ \text{paperID } fpc) \end{aligned}$$
shows $uIsInvar \ \varphi$
 $\langle \text{proof} \rangle$

lemma $uuIsInvar[\text{case-names } uuNews \ uuDis \ uuReview \ uuDec]:$

assumes

$\bigwedge s \ \text{confID } uID \ p \ \text{comm}.$

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

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

and

$\bigwedge s \ \text{confID } uID \ p \ \text{paperID } \text{comm}.$

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

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

and

$\bigwedge s \ \text{confID } uID \ p \ \text{paperID } n \ \text{rc}.$

$\llbracket \text{reach } s; \varphi \ s; \text{e-uupdateReview } s \ \text{confID } uID \ p \ \text{paperID } n \ \text{rc} \rrbracket$

$\implies \varphi \ (\text{uupdateReview } s \ \text{confID } uID \ p \ \text{paperID } n \ \text{rc})$

and

$\bigwedge s \ \text{confID } uID \ p \ \text{paperID } \text{decision}.$

$\llbracket \text{reach } s; \varphi \ s; \text{e-uupdateDec } s \ \text{confID } uID \ p \ \text{paperID } \text{decision} \rrbracket$

$\implies \varphi \ (\text{uupdateDec } s \ \text{confID } uID \ p \ \text{paperID } \text{decision})$

shows $uuIsInvar \ \varphi$

$\langle \text{proof} \rangle$

3.2 Safety proofs

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

fun papIDsOfRole **where**

$\text{papIDsOfRole} \ (\text{Aut } \text{papID}) = [\text{papID}]$

|

$\text{papIDsOfRole} \ (\text{Rev } \text{papID } n) = [\text{papID}]$

|

$\text{papIDsOfRole} \ - = []$

definition $\text{phase-leq-closedPH} :: \text{state} \Rightarrow \text{bool}$ **where**

$\text{phase-leq-closedPH } s \equiv$

$\forall \ \text{confID}. \ \text{phase } s \ \text{confID} \leq \text{closedPH}$

lemma $\text{holdsIstate-phase-leq-closedPH}: \text{holdsIstate } \text{phase-leq-closedPH}$

$\langle \text{proof} \rangle$

lemma $\text{cIsInvar-phase-leq-closedPH}: \text{cIsInvar } \text{phase-leq-closedPH}$

$\langle \text{proof} \rangle$

lemma *uIsInvar-phase-leq-closedPH*: *uIsInvar phase-leq-closedPH*
<proof>

lemma *uuIsInvar-phase-leq-closedPH*: *uuIsInvar phase-leq-closedPH*
<proof>

lemma *invar-phase-leq-closedPH*: *invar phase-leq-closedPH*
<proof>

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 ≤ closedPH*
<proof>

definition *geq-noPH-confIDs* :: *state ⇒ bool* **where**
geq-noPH-confIDs s ≡
 $\forall \text{ confID. phase } s \text{ confID} > \text{noPH} \longrightarrow \text{confID} \in \in \text{confIDs } s$

lemma *holdsIstate-geq-noPH-confIDs*: *holdsIstate geq-noPH-confIDs*
<proof>

lemma *cIsInvar-geq-noPH-confIDs*: *cIsInvar geq-noPH-confIDs*
<proof>

lemma *uIsInvar-geq-noPH-confIDs*: *uIsInvar geq-noPH-confIDs*
<proof>

lemma *uuIsInvar-geq-noPH-confIDs*: *uuIsInvar geq-noPH-confIDs*
<proof>

lemma *invar-geq-noPH-confIDs*: *invar geq-noPH-confIDs*
<proof>

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 ∈ ∈ confIDs s*
<proof>

definition *roles-IDsOK* :: *state ⇒ bool* **where**
roles-IDsOK s ≡

$\forall \text{ confID } uID \text{ } rl.$
 $rl \in \in \text{ roles } s \text{ confID } uID \longrightarrow \text{ IDsOK } s [\text{ confID}] [uID] (\text{ papIDsOfRole } rl)$

lemma *holdsIstate-roles-IDsOK*: *holdsIstate roles-IDsOK*
 $\langle \text{proof} \rangle$

lemma *cIsInvar-roles-IDsOK*: *cIsInvar roles-IDsOK*
 $\langle \text{proof} \rangle$

lemma *uIsInvar-roles-IDsOK*: *uIsInvar roles-IDsOK*
 $\langle \text{proof} \rangle$

lemma *uuIsInvar-roles-IDsOK*: *uuIsInvar roles-IDsOK*
 $\langle \text{proof} \rangle$

lemma *invar-roles-IDsOK*: *invar roles-IDsOK*
 $\langle \text{proof} \rangle$

lemmas *roles-IDsOK1* =
 $\text{holdsIstate-invar}[\text{OF } \text{holdsIstate-roles-IDsOK } \text{invar-roles-IDsOK}]$

theorem *roles-IDsOK*:
assumes *a*: *reach s* **and** *rl*: $rl \in \in \text{ roles } s \text{ confID } uID$
shows $\text{ IDsOK } s [\text{ confID}] [uID] (\text{ papIDsOfRole } rl)$
 $\langle \text{proof} \rangle$

corollary *roles-confIDs*:
assumes *a*: *reach s* **and** *A*: $rl \in \in \text{ roles } s \text{ confID } uID$
shows $\text{ confID } \in \in \text{ confIDs } s$
 $\langle \text{proof} \rangle$

corollary *roles-userIDs*:
assumes *a*: *reach s* **and** *A*: $rl \in \in \text{ roles } s \text{ confID } uID$
shows $uID \in \in \text{ userIDs } s$
 $\langle \text{proof} \rangle$

corollary *isAut-paperIDs*:
assumes *a*: *reach s* **and** *A*: $\text{ isAut } s \text{ confID } uID \text{ papID}$
shows $\text{ papID } \in \in \text{ paperIDs } s \text{ confID}$
 $\langle \text{proof} \rangle$

corollary *isRevNth-paperIDs*:
assumes *a*: *reach s* **and** *A*: $\text{ isRevNth } s \text{ confID } uID \text{ papID } n$
shows $\text{ papID } \in \in \text{ paperIDs } s \text{ confID}$
 $\langle \text{proof} \rangle$

corollary *isRev-paperIDs*:
assumes *a*: *reach s* **and** *A*: $\text{ isRev } s \text{ confID } uID \text{ papID}$
shows $\text{ papID } \in \in \text{ paperIDs } s \text{ confID}$

<proof>

corollary *isRev-userIDs:*

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

shows *uID ∈ userIDs s*

<proof>

corollary *isRev-confIDs:*

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

shows *confID ∈ confIDs s*

<proof>

definition *distinct-IDs :: state ⇒ bool where*

distinct-IDs s ≡

distinct (confIDs s) ∧ distinct (userIDs s) ∧ (∀ confID. distinct (paperIDs s confID))

lemma *holdsIstate-distinct-IDs: holdsIstate distinct-IDs*

<proof>

lemma *cIsInvar-distinct-IDs: cIsInvar distinct-IDs*

<proof>

lemma *uIsInvar-distinct-IDs: uIsInvar distinct-IDs*

<proof>

lemma *uuIsInvar-distinct-IDs: uuIsInvar distinct-IDs*

<proof>

lemma *invar-distinct-IDs: invar distinct-IDs*

<proof>

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

theorem *distinct-IDs:*

assumes *a: reach s*

shows *distinct (confIDs s) ∧ distinct (userIDs s) ∧ (∀ confID. distinct (paperIDs s confID))*

<proof>

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 ⇒ bool where*

distinct-roles s ≡

∀ confID uID. distinct (roles s confID uID)

lemma *holdsIstate-distinct-roles: holdsIstate distinct-roles*
{proof}

lemma *cIsInvar-distinct-roles: cIsInvar distinct-roles*
{proof}

lemma *uIsInvar-distinct-roles: uIsInvar distinct-roles*
{proof}

lemma *uuIsInvar-distinct-roles: uuIsInvar distinct-roles*
{proof}

lemma *invar-distinct-roles: invar distinct-roles*
{proof}

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)*
{proof}

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*
{proof}

lemma *cIsInvar-isRevNth-isPC: cIsInvar isRevNth-isPC*
{proof}

lemma *uIsInvar-isRevNth-isPC: uIsInvar isRevNth-isPC*
{proof}

lemma *uuIsInvar-isRevNth-isPC: uuIsInvar isRevNth-isPC*
{proof}

lemma *invar-isRevNth-isPC: invar isRevNth-isPC*
{proof}

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*
{proof}

corollary *isRev-isPC*:

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

shows *isPC* *s* *confID* *uID*

<proof>

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*

<proof>

lemma *cIsInvar-paperIDs-confIDs*: *cIsInvar* *paperIDs-confIDs*

<proof>

lemma *uIsInvar-paperIDs-confIDs*: *uIsInvar* *paperIDs-confIDs*

<proof>

lemma *uuIsInvar-paperIDs-confIDs*: *uuIsInvar* *paperIDs-confIDs*

<proof>

lemma *invar-paperIDs-confIDs*: *invar* *paperIDs-confIDs*

<proof>

lemmas *paperIDs-confIDs*1 = *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*

<proof>

corollary *paperIDs-getAllPaperIDs*:

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

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

<proof>

corollary *isRevNth-getAllPaperIDs*:

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

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

<proof>

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

paperIDs-equals *s* \equiv

\forall *confID*1 *confID*2 *papID*.

$papID \in \in paperIDs\ s\ confID1 \wedge papID \in \in paperIDs\ s\ confID2$
 $\longrightarrow confID1 = confID2$

lemma *holdsIstate-paperIDs-equals: holdsIstate paperIDs-equals*
 $\langle proof \rangle$

lemma *cIsInvar-paperIDs-equals: cIsInvar paperIDs-equals*
 $\langle proof \rangle$

lemma *uIsInvar-paperIDs-equals: uIsInvar paperIDs-equals*
 $\langle proof \rangle$

lemma *uuIsInvar-paperIDs-equals: uuIsInvar paperIDs-equals*
 $\langle proof \rangle$

lemma *invar-paperIDs-equals: invar paperIDs-equals*
 $\langle proof \rangle$

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

theorem *paperIDs-equals:*

assumes *a: reach s and p: papID ∈ ∈ paperIDs s confID1 papID ∈ ∈ paperIDs s confID2*

shows *confID1 = confID2*
 $\langle proof \rangle$

definition *isAut-pref-Conflict :: state ⇒ bool where*

isAut-pref-Conflict s ≡

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

lemma *holdsIstate-isAut-pref-Conflict: holdsIstate isAut-pref-Conflict*
 $\langle proof \rangle$

lemma *cIsInvar-isAut-pref-Conflict: cIsInvar isAut-pref-Conflict*
 $\langle proof \rangle$

lemma *uIsInvar-isAut-pref-Conflict: uIsInvar isAut-pref-Conflict*
 $\langle proof \rangle$

lemma *uuIsInvar-isAut-pref-Conflict: uuIsInvar isAut-pref-Conflict*
 $\langle proof \rangle$

lemma *invar-isAut-pref-Conflict: invar isAut-pref-Conflict*
 $\langle proof \rangle$

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*
⟨*proof*⟩

definition *phase-noPH-paperIDs* :: state ⇒ bool **where**
phase-noPH-paperIDs *s* ≡
∀ *confID*. *phase s confID = noPH* → *paperIDs s confID = []*

lemma *holdsIstate-phase-noPH-paperIDs*: holdsIstate *phase-noPH-paperIDs*
⟨*proof*⟩

lemma *cIsInvar-phase-noPH-paperIDs*: *cIsInvar phase-noPH-paperIDs*
⟨*proof*⟩

lemma *uIsInvar-phase-noPH-paperIDs*: *uIsInvar phase-noPH-paperIDs*
⟨*proof*⟩

lemma *uuIsInvar-phase-noPH-paperIDs*: *uuIsInvar phase-noPH-paperIDs*
⟨*proof*⟩

lemma *invar-phase-noPH-paperIDs*: *invar phase-noPH-paperIDs*
⟨*proof*⟩

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 = []*
⟨*proof*⟩

definition *paperIDs-geq-subPH* :: state ⇒ bool **where**
paperIDs-geq-subPH *s* ≡
∀ *confID* *papID*. *papID* ∈ *paperIDs s confID* → *phase s confID* ≥ *subPH*

lemma *holdsIstate-paperIDs-geq-subPH*: holdsIstate *paperIDs-geq-subPH*
⟨*proof*⟩

lemma *cIsInvar-paperIDs-geq-subPH*: *cIsInvar paperIDs-geq-subPH*
⟨*proof*⟩

lemma *uIsInvar-paperIDs-geq-subPH*: *uIsInvar paperIDs-geq-subPH*
⟨*proof*⟩

lemma *uuIsInvar-paperIDs-geq-subPH*: *uuIsInvar paperIDs-geq-subPH*

<proof>

lemma *invar-paperIDs-geq-subPH*: *invar paperIDs-geq-subPH*
<proof>

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*: *paperID ∈∈ paperIDs s confID*
shows *phase s confID ≥ subPH*
<proof>

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

lemma *holdsIstate-isRevNth-geq-revPH*: *holdsIstate isRevNth-geq-revPH*
<proof>

lemma *cIsInvar-isRevNth-geq-revPH*: *cIsInvar isRevNth-geq-revPH*
<proof>

lemma *uIsInvar-isRevNth-geq-revPH*: *uIsInvar isRevNth-geq-revPH*
<proof>

lemma *uuIsInvar-isRevNth-geq-revPH*: *uuIsInvar isRevNth-geq-revPH*
<proof>

lemma *invar-isRevNth-geq-revPH*: *invar isRevNth-geq-revPH*
<proof>

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 paperID n*
shows *phase s confID ≥ revPH*
<proof>

corollary *isRev-geq-revPH*:
assumes *a*: *reach s* **and** *i*: *isRev s confID uID paperID*
shows *phase s confID ≥ revPH*
<proof>

definition *paperID-ex-userID* :: *state ⇒ bool* **where**

paperID-ex-userID $s \equiv$
 $\forall \text{ confID } \text{papID}. \text{papID} \in \in \text{paperIDs } s \text{ confID} \longrightarrow (\exists \text{ uID}. \text{isAut } s \text{ confID } \text{uID } \text{papID})$

lemma *holdsIstate-paperID-ex-userID*: *holdsIstate paperID-ex-userID*
 $\langle \text{proof} \rangle$

lemma *cIsInvar-paperID-ex-userID*: *cIsInvar paperID-ex-userID*
 $\langle \text{proof} \rangle$

lemma *uIsInvar-paperID-ex-userID*: *uIsInvar paperID-ex-userID*
 $\langle \text{proof} \rangle$

lemma *uuIsInvar-paperID-ex-userID*: *uuIsInvar paperID-ex-userID*
 $\langle \text{proof} \rangle$

lemma *invar-paperID-ex-userID*: *invar paperID-ex-userID*
 $\langle \text{proof} \rangle$

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* $\in \in$ *paperIDs s confID*
shows $\exists \text{ uID}. \text{isAut } s \text{ confID } \text{uID } \text{papID}$
 $\langle \text{proof} \rangle$

definition *pref-Conflict-isRevNth* :: *state* \Rightarrow *bool* **where**
pref-Conflict-isRevNth $s \equiv$
 $\forall \text{ confID } \text{uID } \text{papID } n. \text{pref } s \text{ uID } \text{papID} = \text{Conflict} \longrightarrow \neg \text{isRevNth } s \text{ confID } \text{uID } \text{papID } n$

lemma *holdsIstate-pref-Conflict-isRevNth*: *holdsIstate pref-Conflict-isRevNth*
 $\langle \text{proof} \rangle$

lemma *cIsInvar-pref-Conflict-isRevNth*: *cIsInvar pref-Conflict-isRevNth*
 $\langle \text{proof} \rangle$

lemma *uIsInvar-pref-Conflict-isRevNth*: *uIsInvar pref-Conflict-isRevNth*
 $\langle \text{proof} \rangle$

lemma *uuIsInvar-pref-Conflict-isRevNth*: *uuIsInvar pref-Conflict-isRevNth*
 $\langle \text{proof} \rangle$

lemma *invar-pref-Conflict-isRevNth*: *invar pref-Conflict-isRevNth*
 $\langle \text{proof} \rangle$

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 uID papID = Conflict*

shows \neg *isRevNth s confID uID papID n*

<proof>

corollary *pref-Conflict-isRev*:

assumes *a*: *reach s* **and** *i*: *pref s uID papID = Conflict*

shows \neg *isRev s confID uID papID*

<proof>

corollary *pref-isAut-isRevNth*:

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

shows \neg *isRevNth s confID uID papID n*

<proof>

corollary *pref-isAut-isRev*:

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

shows \neg *isRev s confID uID papID*

<proof>

definition *isChair-isPC* :: *state* \Rightarrow *bool* **where**

isChair-isPC s \equiv

\forall *confID uID*. *isChair s confID uID* \longrightarrow *isPC s confID uID*

lemma *holdsIstate-isChair-isPC*: *holdsIstate isChair-isPC*

<proof>

lemma *cIsInvar-isChair-isPC*: *cIsInvar isChair-isPC*

<proof>

lemma *uIsInvar-isChair-isPC*: *uIsInvar isChair-isPC*

<proof>

lemma *uuIsInvar-isChair-isPC*: *uuIsInvar isChair-isPC*

<proof>

lemma *invar-isChair-isPC*: *invar isChair-isPC*

<proof>

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*

<proof>

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*

<proof>

lemma *cIsInvar-isRevNth-equals*: *cIsInvar isRevNth-equals*

<proof>

lemma *uIsInvar-isRevNth-equals*: *uIsInvar isRevNth-equals*

<proof>

lemma *uuIsInvar-isRevNth-equals*: *uuIsInvar isRevNth-equals*

<proof>

lemma *invar-isRevNth-equals*: *invar isRevNth-equals*

<proof>

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*

<proof>

corollary *isRevNth-getReviewIndex*:

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

shows *n = getReviewIndex s confID uID papID*

<proof>

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*

<proof>

lemma *cIsInvar-isRevNth-less-length*: *cIsInvar isRevNth-less-length*

<proof>

lemma *uIsInvar-isRevNth-less-length*: *uIsInvar isRevNth-less-length*
<proof>

lemma *uuIsInvar-isRevNth-less-length*: *uuIsInvar isRevNth-less-length*
<proof>

lemma *invar-isRevNth-less-length*: *invar isRevNth-less-length*
<proof>

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))*
<proof>

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*
<proof>

lemma *cIsInvar-isRevNth-equalsU*: *cIsInvar isRevNth-equalsU*
<proof>

lemma *uIsInvar-isRevNth-equalsU*: *uIsInvar isRevNth-equalsU*
<proof>

lemma *uuIsInvar-isRevNth-equalsU*: *uuIsInvar isRevNth-equalsU*
<proof>

lemma *invar-isRevNth-equalsU*: *invar isRevNth-equalsU*
<proof>

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*

<proof>

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*

<proof>

lemma *cIsInvar-reviews-compact*: *cIsInvar* *reviews-compact*

<proof>

lemma *uIsInvar-reviews-compact*: *uIsInvar* *reviews-compact*

<proof>

lemma *uuIsInvar-reviews-compact*: *uuIsInvar* *reviews-compact*

<proof>

lemma *invar-reviews-compact*: *invar* *reviews-compact*

<proof>

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 $\exists\ uid.\ isRevNth\ s\ cid\ uid\ pid\ n$

<proof>

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*

<proof>

lemma *cIsInvar-roles-nonrep*: *cIsInvar* *roles-nonrep*

<proof>

lemma *uIsInvar-roles-nonrep*: *uIsInvar* *roles-nonrep*

<proof>

lemma *uuIsInvar-roles-nonrep*: *uuIsInvar roles-nonrep*
(proof)

lemma *invar-roles-nonrep*: *invar roles-nonrep*
(proof)

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

theorem *roles-nonrep*:
assumes *reach s*
shows *distinct (roles s confID uID)*
(proof)

3.3 Properties of the step function

lemma *step-outErr-eq*: *step s a = (outErr, s') \implies s' = s*
(proof)

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

lemma *phase-increases2*: *phase s CID \leq phase (snd (step s a)) CID*
(proof)

lemma *confIDs-mono*:
assumes *step s a = (ou, s')* **and** *cid $\in\in$ confIDs s*
shows *cid $\in\in$ confIDs s'*
(proof)

lemma *userIDs-mono*:
assumes *step s a = (ou, s')* **and** *uid $\in\in$ userIDs s*
shows *uid $\in\in$ userIDs s'*
(proof)

lemma *paperIDs-mono*:
assumes *step s a = (ou, s')* **and** *pid $\in\in$ paperIDs s cid*
shows *pid $\in\in$ paperIDs s' cid*
(proof)

lemma *isPC-persistent*:
assumes *isPC s cid uid* **and** *step s a = (ou, s')*
shows *isPC s' cid uid*
(proof)

lemma *isChair-persistent*:
assumes *isChair s cid uid* **and** *step s a = (ou, s')*

$\wedge uu. s \langle paperIDs := uu, paper := pp \rangle = s \langle paper := pp, paperIDs := uu \rangle$
 $\wedge uu. s \langle pref := uu, paper := pp \rangle = s \langle paper := pp, pref := uu \rangle$
 $\wedge uu. s \langle voronkov := uu, paper := pp \rangle = s \langle paper := pp, voronkov := uu \rangle$
 $\wedge uu. s \langle news := uu, paper := pp \rangle = s \langle paper := pp, news := uu \rangle$
 $\wedge uu. s \langle phase := uu, paper := pp \rangle = s \langle paper := pp, phase := uu \rangle$
 $\langle proof \rangle$

lemma *isAUT-imp-isAut*:
assumes *reach s and pid ∈∈ paperIDs s cid and isAUT s uid pid*
shows *isAut s cid uid pid*
 $\langle proof \rangle$

lemma *isREVNth-imp-isRevNth*:
assumes *reach s and pid ∈∈ paperIDs s cid and isREVNth s uid pid n*
shows *isRevNth s cid uid pid n*
 $\langle proof \rangle$

lemma *phase-increases-validTrans*:
assumes *validTrans (Trans s a ou s')*
shows *phase s cid ≤ phase s' cid*
 $\langle proof \rangle$

lemma *phase-increases-validTrans2*:
assumes *validTrans tr*
shows *phase (srcOf tr) cid ≤ phase (tgtOf tr) cid*
 $\langle proof \rangle$

lemma *phase-increases-trace*:
assumes *vtr: valid tr and ij: i ≤ j and j: j < length tr*
shows *phase (srcOf (tr!i)) cid ≤ phase (srcOf (tr!j)) cid*
 $\langle proof \rangle$

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*
 $\langle proof \rangle$

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*
 $\langle proof \rangle$

lemma *phase-increases-trace-srcOf-last*:
assumes v : valid tr **and** l : length $tr > 1$ **and** i : $i < \text{length } tr$
shows $\text{phase } (\text{srcOf } (tr!i)) \text{ cid} \leq \text{phase } (\text{srcOf } (\text{last } tr)) \text{ cid}$
 $\langle \text{proof} \rangle$

lemma *phase-increases-trace-srcOf-tgtOf-last*:
assumes v : valid tr **and** l : length $tr > 1$ **and** i : $i < \text{length } tr$
shows $\text{phase } (\text{srcOf } (tr!i)) \text{ cid} \leq \text{phase } (\text{tgtOf } (\text{last } tr)) \text{ cid}$
 $\langle \text{proof} \rangle$

lemma *valid-tgtPf-last-srcOf*:
assumes valid tr **and** $s \in \in \text{map } \text{tgtOf } tr$
shows $s = \text{tgtOf } (\text{last } tr) \vee s \in \in \text{map } \text{srcOf } tr$
 $\langle \text{proof} \rangle$

lemma *phase-constant*:
assumes v : valid tr **and** l : length $tr > 0$ **and**
 ph : $\text{phase } (\text{srcOf } (\text{hd } tr)) \text{ cid} = \text{phase } (\text{tgtOf } (\text{last } tr)) \text{ cid}$
shows $\text{set } (\text{map } (\lambda \text{trn}. \text{phase } (\text{srcOf } \text{trn}) \text{ cid}) tr) \subseteq \{\text{phase } (\text{srcOf } (\text{hd } tr)) \text{ cid}\}$
 \wedge
 $\text{set } (\text{map } (\lambda \text{trn}. \text{phase } (\text{tgtOf } \text{trn}) \text{ cid}) tr) \subseteq \{\text{phase } (\text{srcOf } (\text{hd } tr)) \text{ cid}\}$
 $\langle \text{proof} \rangle$

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

$\mid (Id) \ \text{phase } s' \ \text{cid} = \text{phase } s \ \text{cid}$
 $\mid (Upd) \ \text{uid } p \ \text{ph} \ \mathbf{where} \ \text{phase } s' \ \text{cid} = \text{ph } a = Uact \ (uPhase \ \text{cid} \ \text{uid } p \ \text{ph})$
 $e\text{-updatePhase } s \ \text{cid} \ \text{uid } p \ \text{ph}$
 $\langle \text{proof} \rangle$

lemma *phase-mono*: $\text{reachFrom } s \ s' \implies \text{phase } s \ \text{cid} \leq \text{phase } s' \ \text{cid}$
 $\langle \text{proof} \rangle$

lemma *validTrans-rAct-lAct-srcOf-tgtOf*:
assumes validTrans trn
and $\text{actOf } trn = Ract \ rAct \vee \text{actOf } trn = Lact \ lAct$
shows $\text{tgtOf } trn = \text{srcOf } trn$
 $\langle \text{proof} \rangle$

lemma *valid-rAct-lAct-srcOf-tgtOf*:
assumes valid tr
and $\bigwedge a. a \in \in \text{map } \text{actOf } tr \implies (\exists rAct. a = Ract \ rAct) \vee (\exists lAct. a = Lact \ lAct)$
shows $\text{srcOf } ' (\text{set } tr) \subseteq \{\text{srcOf } (\text{hd } tr)\}$
 $\langle \text{proof} \rangle$

lemma *validFrom-rAct-lAct-srcOf-tgtOf*:

assumes *validFrom s tr*
and $\bigwedge a. a \in \text{map actOf } tr \implies (\exists rAct. a = Ract\ rAct) \vee (\exists lAct. a = Lact\ lAct)$
shows $\text{srcOf } (set\ tr) \subseteq \{s\}$
 $\langle proof \rangle$

lemma *tgtOf-last-traceOf-Ract-Lact[simp]*:
assumes $al \neq []$ $set\ al \subseteq \text{range } Ract \cup \text{range } Lact$
shows $\text{tgtOf } (\text{last } (\text{traceOf } s\ al)) = s$
 $\langle proof \rangle$

lemma *paperIDs-cases*:
assumes $step\ s\ a = (ou, s')$
obtains $(Id)\ \text{paperIDs } s'\ cid = \text{paperIDs } s\ cid$
 $| (Create)\ cid\ uid\ p\ pid\ tit\ ab$ **where**
 $\text{paperIDs } s'\ cid = pid \# \text{paperIDs } s\ cid$ $a = Cact\ (cPaper\ cid\ uid\ p\ pid\ tit\ ab)$
 $e\text{-createPaper } s\ cid\ uid\ p\ pid\ tit\ ab$
 $\langle proof \rangle$

lemma *paperIDs-decPH-const*:
assumes $s: step\ s\ a = (ou, s')$ **and** $phase\ s\ cid > subPH$
shows $\text{paperIDs } s'\ cid = \text{paperIDs } s\ cid$
 $\langle proof \rangle$

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

4 Observation setup for confidentiality properties

The observation infrastructure, consisting of a discriminator γ and a selector g , is the same for all our confidentiality properties. Namely, we fix a group 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 ⇒ paper ⇒ bool where
  eqButC (Paper name info ct reviews dis decs )
    (Paper name1 info1 ct1 reviews1 dis1 decs1) =
    (name = name1 ∧ info = info1 ∧ reviews = reviews1 ∧ dis = dis1 ∧ decs =
    decs1)

```

lemma *eqButC*:

eqButC pap pap1 =

*(titlePaper pap = titlePaper pap1 ∧ abstractPaper pap = abstractPaper pap1 ∧
reviewsPaper pap = reviewsPaper pap1 ∧ disPaper pap = disPaper pap1 ∧ decsPaper pap = decsPaper pap1)*

<proof>

lemma *eqButC-eq[simp,intro!]*: *eqButC pap pap*

<proof>

lemma *eqButC-sym*:

assumes *eqButC pap pap1*

shows *eqButC pap1 pap*

<proof>

lemma *eqButC-trans*:

assumes *eqButC pap pap1* **and** *eqButC pap1 pap2*

shows *eqButC pap pap2*

<proof>

definition *eeqButPID* **where**

eeqButPID paps paps1 ≡

∀ pid. if pid = PID then eqButC (paps pid) (paps1 pid) else paps pid = paps1 pid

lemma *eeqButPID-eeq[simp,intro!]*: *eeqButPID s s*

<proof>

lemma *eeqButPID-sym*:

assumes *eeqButPID s s1* **shows** *eeqButPID s1 s*

<proof>

lemma *eeqButPID-trans*:

assumes *eeqButPID s s1* **and** *eeqButPID s1 s2* **shows** *eeqButPID s s2*

<proof>

lemma *eeqButPID-imp*:

eeqButPID paps paps1 ⇒ eqButC (paps PID) (paps1 PID)

[[eeqButPID paps paps1; pid ≠ PID]] ⇒ paps pid = paps1 pid

<proof>

lemma *eeqButPID-cong*:

assumes *eeqButPID paps paps1*

and *pid = PID ⇒ eqButC uu uu1*

and *pid ≠ PID ⇒ uu = uu1*

shows *eeqButPID (paps (pid := uu)) (paps1 (pid := uu1))*

<proof>

lemma *eeqButPID-RDD*:

eeqButPID 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)
\langle proof \rangle

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
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

lemma *eqButPID-eq[simp,intro!]*: *eqButPID s s*
\langle proof \rangle

lemma *eqButPID-sym*:

assumes *eqButPID s s1* **shows** *eqButPID s1 s*
\langle proof \rangle

lemma *eqButPID-trans*:

assumes *eqButPID s s1* **and** *eqButPID s1 s2* **shows** *eqButPID s s2*
\langle proof \rangle

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

$isRev\ s\ cid\ uid\ pid = isRev\ s1\ cid\ uid\ pid \wedge$
 $getReviewIndex\ s\ cid\ uid\ pid = getReviewIndex\ s1\ cid\ uid\ pid \wedge$
 $getRevRole\ s\ cid\ uid\ pid = getRevRole\ s1\ cid\ uid\ pid$
 <proof>

lemma *eqButPID-imp1*:

$eqButPID\ s\ s1 \implies eqButC\ (paper\ s\ pid)\ (paper\ s1\ pid)$
 $eqButPID\ s\ s1 \implies pid \neq PID \vee PID \neq pid \implies$
 $\quad paper\ s\ pid = paper\ s1\ pid \wedge$
 $\quad getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$
 <proof>

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>

lemma *eqButPID-RDD*:

$eqButPID\ s\ s1 \implies$
 $titlePaper\ (paper\ s\ PID) = titlePaper\ (paper\ s1\ PID) \wedge$
 $abstractPaper\ (paper\ s\ PID) = abstractPaper\ (paper\ s1\ PID) \wedge$
 $reviewsPaper\ (paper\ s\ PID) = reviewsPaper\ (paper\ s1\ PID) \wedge$
 $disPaper\ (paper\ s\ PID) = disPaper\ (paper\ s1\ PID) \wedge$
 $decsPaper\ (paper\ s\ PID) = decsPaper\ (paper\ s1\ PID)$
 <proof>

lemma *eqButPID-cong[simp, intro]*:

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|confIDs := uu1|))$
 $(s1\ (\!|confIDs := uu2|))$
 $\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|conf := uu1|))\ (s1$
 $(\!|conf := uu2|))$

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|userIDs := uu1|))$
 $(s1\ (\!|userIDs := uu2|))$

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|pass := uu1|))\ (s1$
 $(\!|pass := uu2|))$

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|user := uu1|))\ (s1$
 $(\!|user := uu2|))$

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|roles := uu1|))\ (s1$
 $(\!|roles := uu2|))$

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|paperIDs := uu1|))$
 $(s1\ (\!|paperIDs := uu2|))$

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies eqButPID\ uu1\ uu2 \implies eqButPID\ (s\ (\!|paper :=$
 $uu1|))\ (s1\ (\!|paper := uu2|))$

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|pref := uu1|))\ (s1$
 $(\!|pref := uu2|))$

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|voronkov := uu1|))$

$(s1 \langle \text{voronkov} := uu2 \rangle)$
 $\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s \langle \text{news} := uu1 \rangle) (s1 \langle \text{news} := uu2 \rangle)$
 $\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s \langle \text{phase} := uu1 \rangle) (s1 \langle \text{phase} := uu2 \rangle)$

$\langle \text{proof} \rangle$

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 $\text{title} = \text{title1} \wedge \text{abstract} = \text{abstract1} \wedge \text{reviews} = \text{reviews1} \wedge \text{dis} = \text{dis1} \wedge \text{decs} = \text{decs1}$

$\langle \text{proof} \rangle$

definition *NOSIMP a* $\equiv a$

lemma [*cong*]: *NOSIMP a = NOSIMP a* $\langle \text{proof} \rangle$

lemma *eqButPID-paper*:

assumes *eqButPID s s1*

shows *paper s = (paper s1)(PID :=*

Paper (titlePaper (paper s1 PID))

(abstractPaper (paper s1 PID))

(contentPaper (NOSIMP (paper s PID)))

(reviewsPaper (paper s1 PID))

(disPaper (paper s1 PID))

(decsPaper (paper s1 PID))

)

$\langle \text{proof} \rangle$

lemmas *eqButPID-simps = eqButPID-imp eqButPID-paper*

5.2 Value Setup

type-synonym *value = pcontent*

fun $\varphi :: (\text{state}, \text{act}, \text{out}) \text{ trans} \Rightarrow \text{bool}$ **where**

$\varphi (\text{Trans} - (\text{Uact} (\text{uPaperC cid uid p pid ct})) \text{ ou} -) = (\text{pid} = \text{PID} \wedge \text{ou} = \text{outOK})$

|

$\varphi - = \text{False}$

lemma φ -def2:

$\varphi (\text{Trans} s a \text{ ou} s') = (\exists \text{ cid uid p ct. } a = \text{Uact} (\text{uPaperC cid uid p PID ct}) \wedge \text{ou} = \text{outOK})$

$\langle \text{proof} \rangle$

fun $f :: (\text{state}, \text{act}, \text{out}) \text{ trans} \Rightarrow \text{value}$ **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 $eqButPID\ s\ s'$

<proof>

lemma *φ -step-eqButPID*:

assumes $\varphi: \varphi (Trans\ s\ a\ ou\ s')$

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

shows $eqButPID\ s\ s'$

<proof>

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>

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 $\varphi (Trans\ s1\ a\ ou1\ s1')$

<proof>

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 $\varphi (Trans\ s\ a\ ou\ s') = \varphi (Trans\ s1\ a\ ou1\ s1')$

<proof>

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**
 $T (Trans \text{ - - } ou \ s') =$
 $(\exists \text{ uid} \in UIDs.$
 $\text{isAUT } s' \text{ uid } PID \vee$
 $(\exists \text{ cid}. PID \in \in paperIDs \ s' \text{ cid} \wedge \text{isPC } s' \text{ cid } \text{uid} \wedge \text{phase } s' \text{ cid} \geq \text{bidPH})$
 $)$

declare $T.simps$ [$simp \ del$]

definition $B :: value \ list \Rightarrow value \ list \Rightarrow bool$ **where**
 $B \text{ vl } \text{vl1} \equiv \text{vl} \neq []$

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** $B = B$
 $\langle proof \rangle$

lemma $reachNT\text{-non}\text{-isAut}\text{-isPC}\text{-isChair}$:
assumes $reachNT \ s$ **and** $\text{uid} \in UIDs$
shows
 $\neg \text{isAut } s \text{ cid } \text{uid } PID \wedge$
 $(\text{isPC } s \text{ cid } \text{uid} \longrightarrow \neg PID \in \in paperIDs \ s \text{ cid} \vee \text{phase } s \text{ cid} \leq \text{subPH}) \wedge$
 $(\text{isChair } s \text{ cid } \text{uid} \longrightarrow \neg PID \in \in paperIDs \ s \text{ cid} \vee \text{phase } s \text{ cid} \leq \text{subPH})$
 $\langle proof \rangle$

lemma $P\text{-}\varphi\text{-}\gamma$:
assumes $1: reachNT \ s$ **and** $2: step \ s \ a = (ou, s') \ \varphi \ (Trans \ s \ a \ ou \ s')$
shows $\neg \gamma \ (Trans \ s \ a \ ou \ s')$
 $\langle proof \rangle$

major

lemma $eqButPID\text{-step}\text{-out}$:
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 : $PID \in \in paperIDs \ s \ \text{cid}$
and $UIDs$: $userOfA \ a \in UIDs$
shows $ou = ou1$
 $\langle proof \rangle$

definition $\Delta 1 :: state \Rightarrow value \ list \Rightarrow state \Rightarrow value \ list \Rightarrow bool$ **where**
 $\Delta 1 \ s \ \text{vl} \ s1 \ \text{vl1} \equiv \neg (\exists \text{ cid}. PID \in \in paperIDs \ s \ \text{cid}) \wedge s = s1 \wedge B \ \text{vl} \ \text{vl1}$

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 $\Delta 1\ istate\ vl\ istate\ vl1$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta 1$* : *unwind-cont* $\Delta 1\ \{\Delta 1, \Delta 2, \Delta e\}$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta 2$* : *unwind-cont* $\Delta 2\ \{\Delta 2, \Delta 3, \Delta e\}$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta 3$* : *unwind-cont* $\Delta 3\ \{\Delta 3, \Delta e\}$
 $\langle proof \rangle$

definition *K1exit* **where**
 $K1exit\ s \equiv \exists cid. phase\ s\ cid > subPH \wedge PID \in \in paperIDs\ s\ cid$

lemma *invarNT-K1exit*: *invarNT* $K1exit$
 $\langle proof \rangle$

lemma *noVal-K1exit*: *noVal* $K1exit\ v$
 $\langle proof \rangle$

lemma *unwind-exit- Δe* : *unwind-exit* Δe
 $\langle proof \rangle$

theorem *secure*: *secure*
 $\langle proof \rangle$

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 $T :: (state, act, out) \text{ trans} \Rightarrow \text{bool}$ **where**
 $T (Trans \text{ - - } ou \ s') = (\exists \text{ uid} \in \text{UIDs}. \text{isAUT } s' \text{ uid } \text{PID})$

declare $T.simps$ [*simp del*]

definition $B :: \text{value list} \Rightarrow \text{value list} \Rightarrow \text{bool}$ **where**
 $B \text{ vl } \text{vl1} \equiv \text{vl} \neq [] \wedge \text{vl1} \neq [] \wedge \text{last vl} = \text{last vl1}$

interpretation *BD-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** $B = B$
<proof>

lemma *reachNT-non-isAut:*
assumes $\text{reachNT } s$ **and** $\text{uid} \in \text{UIDs}$
shows $\neg \text{isAut } s \text{ cid uid PID}$
<proof>

lemma *T-φ-γ:*
assumes $1: \text{reachNT } s$ **and** $2: \text{step } s \ a = (ou, s') \ \varphi (Trans \ s \ a \ ou \ s')$
shows $\neg \gamma (Trans \ s \ a \ ou \ s')$
<proof>

lemma *eqButPID-step-out:*
assumes $ss1: \text{eqButPID } s \ s1$
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 $\text{PID}: \text{PID} \in \in \text{paperIDs } s \ \text{cid}$
and $\text{ph}: \text{phase } s \ \text{cid} = \text{subPH}$
and $\text{UIDs}: \text{userOfA } a \in \text{UIDs}$
shows $ou = ou1$
<proof>

major

lemma *eqButPID-step-eq:*
assumes $ss1: \text{eqButPID } s \ s1$
and [*simp*]: $a = \text{Uact } (uPaperC \ \text{cid} \ \text{uid} \ p \ \text{PID} \ \text{ct}) \ ou = \text{outOK}$
and $\text{step}: \text{step } s \ a = (ou, s')$ **and** $\text{step1}: \text{step } s1 \ a = (ou', s1')$

shows $s' = s1'$
 $\langle proof \rangle$

definition $\Delta1 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta1\ s\ vl\ s1\ vl1 \equiv$
 $\neg (\exists\ cid.\ PID \in \in paperIDs\ s\ cid) \wedge s = s1 \wedge B\ vl\ vl1$

definition $\Delta2 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta2\ s\ vl\ s1\ vl1 \equiv$
 $(\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid = subPH) \wedge$
 $eqButPID\ s\ s1 \wedge B\ vl\ vl1$

definition $\Delta3 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta3\ s\ vl\ s1\ vl1 \equiv$
 $(\exists\ cid.\ PID \in \in paperIDs\ s\ cid) \wedge s = s1 \wedge vl = [] \wedge vl1 = []$

definition $\Delta e :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta e\ s\ vl\ s1\ vl1 \equiv$
 $(\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > subPH) \wedge vl \neq []$

lemma *istate- $\Delta1$* :
assumes $B: B\ vl\ vl1$
shows $\Delta1\ istate\ vl\ istate\ vl1$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta1$* : *unwind-cont* $\Delta1\ \{\Delta1, \Delta2, \Delta e\}$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta2$* : *unwind-cont* $\Delta2\ \{\Delta2, \Delta3, \Delta e\}$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta3$* : *unwind-cont* $\Delta3\ \{\Delta3, \Delta e\}$
 $\langle proof \rangle$

definition *K1exit* **where**
 $K1exit\ s \equiv \exists\ cid.\ phase\ s\ cid > subPH \wedge PID \in \in paperIDs\ s\ cid$

lemma *invarNT-K1exit*: *invarNT* *K1exit*
 $\langle proof \rangle$

lemma *noVal-K1exit*: *noVal* *K1exit* v
 $\langle proof \rangle$

lemma *unwind-exit- Δe* : *unwind-exit* Δe
 $\langle proof \rangle$

theorem *secure*: *secure*

```

    <proof>

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**

$eqExcNth\ xs\ ys\ n \equiv$
 $length\ xs = length\ ys \wedge (\forall\ i < length\ xs.\ i \neq n \longrightarrow xs!i = ys!i)$

lemma *eqExcNth-eq*[*simp,intro!*]: $eqExcNth\ xs\ xs\ n$
{*proof*}

lemma *eqExcNth-sym*:

assumes $eqExcNth\ xs\ xs1\ n$

shows $eqExcNth\ xs1\ xs\ n$

{*proof*}

lemma *eqExcNth-trans*:

assumes $eqExcNth\ xs\ xs1\ n$ **and** $eqExcNth\ xs1\ xs2\ n$

shows $eqExcNth\ xs\ xs2\ n$

{*proof*}

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*:

$eqExcD\ pap\ pap1 =$

$(titlePaper\ pap = titlePaper\ pap1 \wedge abstractPaper\ pap = abstractPaper\ pap1 \wedge$
 $contentPaper\ pap = contentPaper\ pap1 \wedge$

$disPaper\ pap = disPaper\ pap1 \wedge decsPaper\ pap = decsPaper\ pap1 \wedge$

$eqExcNth\ (reviewsPaper\ pap)\ (reviewsPaper\ pap1)\ N)$

{*proof*}

lemma *eqExcD-eq*[*simp,intro!*]: $eqExcD\ pap\ pap$

{*proof*}

lemma *eqExcD-sym*:

assumes $eqExcD\ pap\ pap1$

shows $eqExcD\ pap1\ pap$

{*proof*}

lemma *eqExcD-trans*:
assumes *eqExcD pap pap1 and eqExcD pap1 pap2*
shows *eqExcD pap pap2*
 ⟨*proof*⟩

definition *eeqExcPID-N where*
eeqExcPID-N paps paps1 \equiv
 $\forall pid. \text{if } pid = PID \text{ then } eqExcD (paps \ pid) (paps1 \ pid) \text{ else } paps \ pid = paps1 \ pid$

lemma *eeqExcPID-N-eeq[simp,intro!]*: *eeqExcPID-N s s*
 ⟨*proof*⟩

lemma *eeqExcPID-N-sym*:
assumes *eeqExcPID-N s s1* **shows** *eeqExcPID-N s1 s*
 ⟨*proof*⟩

lemma *eeqExcPID-N-trans*:
assumes *eeqExcPID-N s s1 and eeqExcPID-N s1 s2* **shows** *eeqExcPID-N s s2*
 ⟨*proof*⟩

lemma *eeqExcPID-N-imp*:
eeqExcPID-N paps paps1 $\implies eqExcD (paps \ PID) (paps1 \ PID)$
 $\llbracket eeqExcPID-N paps paps1; pid \neq PID \rrbracket \implies paps \ pid = paps1 \ pid$
 ⟨*proof*⟩

lemma *eeqExcPID-N-cong*:
assumes *eeqExcPID-N paps paps1*
and *pid = PID $\implies eqExcD uu uu1$*
and *pid \neq PID $\implies uu = uu1$*
shows *eeqExcPID-N (paps (pid := uu)) (paps1 (pid := uu1))*
 ⟨*proof*⟩

lemma *eeqExcPID-N-RDD*:
eeqExcPID-N paps paps1 \implies
 $titlePaper (paps \ PID) = titlePaper (paps1 \ PID) \wedge$
 $abstractPaper (paps \ PID) = abstractPaper (paps1 \ PID) \wedge$
 $contentPaper (paps \ PID) = contentPaper (paps1 \ PID) \wedge$
 $disPaper (paps \ PID) = disPaper (paps1 \ PID) \wedge$
 $decsPaper (paps \ PID) = decsPaper (paps1 \ PID)$
 ⟨*proof*⟩

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
 $eeqExcPID-N\ (paper\ s)\ (paper\ s1)$
 \wedge
 $pref\ s = pref\ s1 \wedge$
 $voronkov\ s = voronkov\ s1 \wedge$
 $news\ s = news\ s1 \wedge phase\ s = phase\ s1$

lemma $eqExcPID-N-eq[simp,intro!]$: $eqExcPID-N\ s\ s$
 $\langle proof \rangle$

lemma $eqExcPID-N-sym$:
assumes $eqExcPID-N\ s\ s1$ **shows** $eqExcPID-N\ s1\ s$
 $\langle proof \rangle$

lemma $eqExcPID-N-trans$:
assumes $eqExcPID-N\ s\ s1$ **and** $eqExcPID-N\ s1\ s2$ **shows** $eqExcPID-N\ s\ s2$
 $\langle proof \rangle$

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
 $pref\ s = pref\ s1 \wedge$
 $voronkov\ s = voronkov\ s1 \wedge$
 $news\ s = news\ s1 \wedge phase\ s = phase\ s1 \wedge$

$getAllPaperIDs\ s = getAllPaperIDs\ s1 \wedge$
 $isRev\ s\ cid\ uid\ pid = isRev\ s1\ cid\ uid\ pid \wedge$
 $getReviewIndex\ s\ cid\ uid\ pid = getReviewIndex\ s1\ cid\ uid\ pid \wedge$
 $getRevRole\ s\ cid\ uid\ pid = getRevRole\ s1\ cid\ uid\ pid \wedge$
 $length\ (reviewsPaper\ (paper\ s\ pid)) = length\ (reviewsPaper\ (paper\ s1\ pid))$
 $\langle proof \rangle$

lemma $eqExcPID-N-imp1$:
 $eqExcPID-N\ s\ s1 \implies eqExcD\ (paper\ s\ pid)\ (paper\ s1\ pid)$
 $eqExcPID-N\ s\ s1 \implies pid \neq PID \vee PID \neq pid \implies$
 $paper\ s\ pid = paper\ s1\ pid \wedge$
 $getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$
 $\langle proof \rangle$

lemma $eqExcPID-N-imp2$:
assumes $eqExcPID-N\ s\ s1$ **and** $pid \neq PID \vee PID \neq pid$

shows $getReviewersReviews\ s\ cid\ pid = getReviewersReviews\ s1\ cid\ pid$
 $\langle proof \rangle$

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
 $getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$
 $\langle proof \rangle$

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 $getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$
 $\langle proof \rangle$

lemma $eqExcPID-N-RDD$:
 $eqExcPID-N\ s\ s1 \implies$
 $titlePaper\ (paper\ s\ PID) = titlePaper\ (paper\ s1\ PID) \wedge$
 $abstractPaper\ (paper\ s\ PID) = abstractPaper\ (paper\ s1\ PID) \wedge$
 $contentPaper\ (paper\ s\ PID) = contentPaper\ (paper\ s1\ PID) \wedge$
 $disPaper\ (paper\ s\ PID) = disPaper\ (paper\ s1\ PID) \wedge$
 $decsPaper\ (paper\ s\ PID) = decsPaper\ (paper\ s1\ PID)$
 $\langle proof \rangle$

lemma $eqExcPID-N-cong[simp, intro]$:
 $\bigwedge uu1\ uu2. eqExcPID-N\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N\ (s\ (\downarrow confIDs := uu1))\ (s1\ (\downarrow confIDs := uu2))$
 $\bigwedge uu1\ uu2. eqExcPID-N\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N\ (s\ (\downarrow conf := uu1))\ (s1\ (\downarrow conf := uu2))$

$\bigwedge uu1\ uu2. eqExcPID-N\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N\ (s\ (\downarrow userIDs := uu1))\ (s1\ (\downarrow userIDs := uu2))$
 $\bigwedge uu1\ uu2. eqExcPID-N\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N\ (s\ (\downarrow pass := uu1))\ (s1\ (\downarrow pass := uu2))$
 $\bigwedge uu1\ uu2. eqExcPID-N\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N\ (s\ (\downarrow user := uu1))\ (s1\ (\downarrow user := uu2))$
 $\bigwedge uu1\ uu2. eqExcPID-N\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N\ (s\ (\downarrow roles := uu1))\ (s1\ (\downarrow roles := uu2))$

$\bigwedge uu1\ uu2. eqExcPID-N\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N\ (s\ (\downarrow paperIDs := uu1))\ (s1\ (\downarrow paperIDs := uu2))$
 $\bigwedge uu1\ uu2. eqExcPID-N\ s\ s1 \implies eqExcPID-N\ uu1\ uu2 \implies eqExcPID-N\ (s\ (\downarrow paper := uu1))\ (s1\ (\downarrow paper := uu2))$

$\bigwedge uu1\ uu2. eqExcPID-N\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N\ (s\ (\downarrow pref := uu1))\ (s1\ (\downarrow pref := uu2))$
 $\bigwedge uu1\ uu2. eqExcPID-N\ s\ s1 \implies uu1 = uu2 \implies eqExcPID-N\ (s\ (\downarrow voronkov := uu1))\ (s1\ (\downarrow voronkov := uu2))$

$uu1)) (s1 (\text{voronkov} := uu2))$
 $\wedge uu1 uu2. \text{eqExcPID-N } s s1 \implies uu1 = uu2 \implies \text{eqExcPID-N } (s (\text{news} := uu1))$
 $(s1 (\text{news} := uu2))$
 $\wedge uu1 uu2. \text{eqExcPID-N } s s1 \implies uu1 = uu2 \implies \text{eqExcPID-N } (s (\text{phase} := uu1))$
 $(s1 (\text{phase} := uu2))$

$\langle \text{proof} \rangle$

lemma *eqExcPID-N-Paper*:

assumes $s's1'$: *eqExcPID-N* $s s1$

and *paper* $s pid = \text{Paper title abstract content reviews dis decs}$

and *paper* $s1 pid = \text{Paper title1 abstract1 content1 reviews1 dis1 decs1}$

shows $\text{title} = \text{title1} \wedge \text{abstract} = \text{abstract1} \wedge \text{content} = \text{content1} \wedge \text{dis} = \text{dis1} \wedge$
 $\text{decs} = \text{decs1}$

$\langle \text{proof} \rangle$

Auxiliary definitions for a slightly weaker equivalence relation defined below:

definition *eqExcNth2* **where**

eqExcNth2 $rl rl1 n \equiv$

$\text{length } rl = \text{length } rl1 \wedge$

$(\forall i < \text{length } rl. i \neq n \longrightarrow rl!i = rl1!i) \wedge$

$\text{hd } (rl!n) = \text{hd } (rl1!n)$

lemma *eqExcNth2-eq[simp,intro]*: *eqExcNth2* $rl rl n$

$\langle \text{proof} \rangle$

lemma *eqExcNth2-sym*:

assumes *eqExcNth2* $rl rl1 n$

shows *eqExcNth2* $rl1 rl n$

$\langle \text{proof} \rangle$

lemma *eqExcNth2-trans*:

assumes *eqExcNth2* $rl rl1 n$ **and** *eqExcNth2* $rl1 rl2 n$

shows *eqExcNth2* $rl rl2 n$

$\langle \text{proof} \rangle$

fun *eqExcD2* :: *paper* \Rightarrow *paper* \Rightarrow *bool* **where**

eqExcD2 (*Paper title abstract ct reviews dis decs*)

(*Paper title1 abstract1 ct1 reviews1 dis1 decs1*) =

$(\text{title} = \text{title1} \wedge \text{abstract} = \text{abstract1} \wedge \text{ct} = \text{ct1} \wedge \text{dis} = \text{dis1} \wedge \text{decs} = \text{decs1} \wedge$
 $\text{eqExcNth2 } \text{reviews} \text{ reviews1 } N)$

lemma *eqExcD2*:

eqExcD2 $\text{pap pap1} =$

$(\text{titlePaper } \text{pap} = \text{titlePaper } \text{pap1} \wedge \text{abstractPaper } \text{pap} = \text{abstractPaper } \text{pap1} \wedge$

$\text{contentPaper } \text{pap} = \text{contentPaper } \text{pap1} \wedge$

$\text{disPaper } \text{pap} = \text{disPaper } \text{pap1} \wedge \text{decsPaper } \text{pap} = \text{decsPaper } \text{pap1} \wedge$

$\text{eqExcNth2 } (\text{reviewsPaper } \text{pap}) (\text{reviewsPaper } \text{pap1}) N)$

$\langle \text{proof} \rangle$

lemma $eqExcD2\text{-}eq[simp,intro!]$: $eqExcD2\ pap\ pap$
 $\langle proof \rangle$

lemma $eqExcD2\text{-}sym$:
assumes $eqExcD2\ pap\ pap1$
shows $eqExcD2\ pap1\ pap$
 $\langle proof \rangle$

lemma $eqExcD2\text{-}trans$:
assumes $eqExcD2\ pap\ pap1$ **and** $eqExcD2\ pap1\ pap2$
shows $eqExcD2\ pap\ pap2$
 $\langle proof \rangle$

definition $eeqExcPID\text{-}N2$ **where**
 $eeqExcPID\text{-}N2\ paps\ paps1 \equiv$
 $\forall\ pid. \text{ if } pid = PID \text{ then } eqExcD2\ (paps\ pid)\ (paps1\ pid) \text{ else } paps\ pid = paps1\ pid$

lemma $eeqExcPID\text{-}N2\text{-}eq[simp,intro!]$: $eeqExcPID\text{-}N2\ s\ s$
 $\langle proof \rangle$

lemma $eeqExcPID\text{-}N2\text{-}sym$:
assumes $eeqExcPID\text{-}N2\ s\ s1$ **shows** $eeqExcPID\text{-}N2\ s1\ s$
 $\langle proof \rangle$

lemma $eeqExcPID\text{-}N2\text{-}trans$:
assumes $eeqExcPID\text{-}N2\ s\ s1$ **and** $eeqExcPID\text{-}N2\ s1\ s2$ **shows** $eeqExcPID\text{-}N2\ s\ s2$
 $\langle proof \rangle$

lemma $eeqExcPID\text{-}N2\text{-}imp$:
 $eeqExcPID\text{-}N2\ paps\ paps1 \implies eqExcD2\ (paps\ PID)\ (paps1\ PID)$
 $\llbracket eeqExcPID\text{-}N2\ paps\ paps1; pid \neq PID \rrbracket \implies paps\ pid = paps1\ pid$
 $\langle proof \rangle$

lemma $eeqExcPID\text{-}N2\text{-}cong$:
assumes $eeqExcPID\text{-}N2\ paps\ paps1$
and $pid = PID \implies eqExcD2\ uu\ uu1$
and $pid \neq PID \implies uu = uu1$
shows $eeqExcPID\text{-}N2\ (paps\ (pid := uu))\ (paps1\ (pid := uu1))$
 $\langle proof \rangle$

lemma $eeqExcPID\text{-}N2\text{-}RDD$:
 $eeqExcPID\text{-}N2\ paps\ paps1 \implies$
 $titlePaper\ (paps\ PID) = titlePaper\ (paps1\ PID) \wedge$
 $abstractPaper\ (paps\ PID) = abstractPaper\ (paps1\ PID) \wedge$
 $contentPaper\ (paps\ PID) = contentPaper\ (paps1\ PID) \wedge$
 $disPaper\ (paps\ PID) = disPaper\ (paps1\ PID) \wedge$
 $decsPaper\ (paps\ PID) = decsPaper\ (paps1\ PID)$

<proof>

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 $eqExcPID-N2-eq[simp,intro!]: eqExcPID-N2\ s\ s$

<proof>

lemma $eqExcPID-N2-sym:$

assumes $eqExcPID-N2\ s\ s1$ **shows** $eqExcPID-N2\ s1\ s$

<proof>

lemma $eqExcPID-N2-trans:$

assumes $eqExcPID-N2\ s\ s1$ **and** $eqExcPID-N2\ s1\ s2$ **shows** $eqExcPID-N2\ s\ s2$

<proof>

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
 $pref\ s = pref\ s1 \wedge$
 $voronkov\ s = voronkov\ s1 \wedge$
 $news\ s = news\ s1 \wedge phase\ s = phase\ s1 \wedge$

$getAllPaperIDs\ s = getAllPaperIDs\ s1 \wedge$
 $isRev\ s\ cid\ uid\ pid = isRev\ s1\ cid\ uid\ pid \wedge$
 $getReviewIndex\ s\ cid\ uid\ pid = getReviewIndex\ s1\ cid\ uid\ pid \wedge$
 $getRevRole\ s\ cid\ uid\ pid = getRevRole\ s1\ cid\ uid\ pid \wedge$
 $length\ (reviewsPaper\ (paper\ s\ pid)) = length\ (reviewsPaper\ (paper\ s1\ pid))$

$\langle \text{proof} \rangle$

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*

$\langle \text{proof} \rangle$

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*

$\langle \text{proof} \rangle$

lemma *eqExcPID-N2-eqExcPID-N*:

eqExcPID-N2 s s1 \implies *eqExcPID-N s s1*

$\langle \text{proof} \rangle$

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*

$\langle \text{proof} \rangle$

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*

$\langle \text{proof} \rangle$

lemma *eqExcPID-N2-imp33*:

assumes *eqExcPID-N2 s s1*

shows *hd (getNthReview s pid N)* = *hd (getNthReview s1 pid N)*

$\langle \text{proof} \rangle$

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)*

$\langle \text{proof} \rangle$

lemma *eqExcPID-N2-cong[simp, intro]*:

$\bigwedge uu1 uu2. eqExcPID-N2 s s1 \implies uu1 = uu2 \implies eqExcPID-N2 (s (\downarrow confIDs := uu1)) (s1 (\downarrow confIDs := uu2))$

$\bigwedge uu1 uu2. eqExcPID-N2 s s1 \implies uu1 = uu2 \implies eqExcPID-N2 (s (\backslash conf := uu1)) (s1 (\backslash conf := uu2))$

$\bigwedge uu1 uu2. eqExcPID-N2 s s1 \implies uu1 = uu2 \implies eqExcPID-N2 (s (\backslash userIDs := uu1)) (s1 (\backslash userIDs := uu2))$

$\bigwedge uu1 uu2. eqExcPID-N2 s s1 \implies uu1 = uu2 \implies eqExcPID-N2 (s (\backslash pass := uu1)) (s1 (\backslash pass := uu2))$

$\bigwedge uu1 uu2. eqExcPID-N2 s s1 \implies uu1 = uu2 \implies eqExcPID-N2 (s (\backslash user := uu1)) (s1 (\backslash user := uu2))$

$\bigwedge uu1 uu2. eqExcPID-N2 s s1 \implies uu1 = uu2 \implies eqExcPID-N2 (s (\backslash roles := uu1)) (s1 (\backslash roles := uu2))$

$\bigwedge uu1 uu2. eqExcPID-N2 s s1 \implies uu1 = uu2 \implies eqExcPID-N2 (s (\backslash paperIDs := uu1)) (s1 (\backslash paperIDs := uu2))$

$\bigwedge uu1 uu2. eqExcPID-N2 s s1 \implies eqExcPID-N2 uu1 uu2 \implies eqExcPID-N2 (s (\backslash paper := uu1)) (s1 (\backslash paper := uu2))$

$\bigwedge uu1 uu2. eqExcPID-N2 s s1 \implies uu1 = uu2 \implies eqExcPID-N2 (s (\backslash pref := uu1)) (s1 (\backslash pref := uu2))$

$\bigwedge uu1 uu2. eqExcPID-N2 s s1 \implies uu1 = uu2 \implies eqExcPID-N2 (s (\backslash voronkov := uu1)) (s1 (\backslash voronkov := uu2))$

$\bigwedge uu1 uu2. eqExcPID-N2 s s1 \implies uu1 = uu2 \implies eqExcPID-N2 (s (\backslash news := uu1)) (s1 (\backslash news := uu2))$

$\bigwedge uu1 uu2. eqExcPID-N2 s s1 \implies uu1 = uu2 \implies eqExcPID-N2 (s (\backslash phase := uu1)) (s1 (\backslash phase := uu2))$

$\langle proof \rangle$

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 $title = title1 \wedge abstract = abstract1 \wedge content = content1 \wedge dis = dis1 \wedge decs = decs1$

$\langle proof \rangle$

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'*

$\langle proof \rangle$

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 φ -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
 $a = UUact (uuReview cid uid p PID N rc)$
 $))$
 $\langle proof \rangle$

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 $eqExcPID$ - $N s s'$
 $\langle proof \rangle$

lemma φ -step- $eqExcPID$ - N :
assumes φ : $\varphi (Trans s a ou s')$
and s : $step s a = (ou, s')$
shows $eqExcPID$ - $N s s'$
 $\langle proof \rangle$

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 $eqExcPID$ - $N s' s1'$
 $\langle proof \rangle$

lemma $eqExcPID$ - N -step- φ -imp:
assumes $ss1$: $eqExcPID$ - $N s s1$
and $step$: $step s a = (ou, s')$ **and** $step1$: $step s1 a = (ou1, s1')$
and φ : $\varphi (Trans s a ou s')$
shows $\varphi (Trans s1 a ou1 s1')$
 $\langle proof \rangle$

lemma $eqExcPID$ - N -step- φ :
assumes $s's1'$: $eqExcPID$ - $N s s1$
and $step$: $step s a = (ou, s')$ **and** $step1$: $step s1 a = (ou1, s1')$
shows $\varphi (Trans s a ou s') = \varphi (Trans s1 a ou1 s1')$

<proof>

lemma *eqExcPID-N2-step-φ-imp*:
assumes *ss1: eqExcPID-N2 s s1*
and *step: step s a = (ou,s[^])* **and** *step1: step s1 a = (ou1,s1[^])*
and *r: N < length (reviewsPaper (paper s PID))*
and *φ: φ (Trans s a ou s[^])*
shows *φ (Trans s1 a ou1 s1[^])*
<proof>

lemma *eqExcPID-N2-step-φ*:
assumes *s: reach s* **and** *s1: reach s1*
and *ss1: eqExcPID-N2 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[^])*
<proof>

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**

$B \text{ vl vl1} \equiv$
 $\exists \text{ ul ul1 wl.}$
 $\text{vl} = (\text{map } (\text{Pair revPH}) \text{ ul}) @ (\text{map } (\text{Pair disPH}) \text{ wl}) \wedge$
 $\text{vl1} = (\text{map } (\text{Pair revPH}) \text{ ul1}) @ (\text{map } (\text{Pair disPH}) \text{ wl}) \wedge$
 $\text{ul} \neq [] \wedge \text{ul1} \neq [] \wedge \text{last ul} = \text{last ul1}$

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 $B = B$

<proof>

lemma *reachNT-non-isRevNth*:

assumes *reachNT s* and *uid* \in *UIDs*

shows $\neg \text{isRevNth } s \text{ cid uid PID } N$

<proof>

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'*)

<proof>

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 *paperIDs s cid*

and *ph*: *phase s cid = revPH*

and *UIDs*: *userOfA a* \in *UIDs*

shows *ou = ou1*

<proof>

lemma *eeqExcPID-N-imp-eq*:

assumes *eeqExcPID-N paps paps1*

and *reviewsPaper (paps PID) ! N = reviewsPaper (paps1 PID) ! N*

shows *paps = paps1*

<proof>

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>

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 $s' = s1'$
 $\langle proof \rangle$

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**
 $\Delta 3\ s\ vl\ s1\ vl1 \equiv$
 $\exists\ cid\ uid.$
 $PID \in\in\ paperIDs\ s\ cid \wedge phase\ s\ cid = revPH \wedge isREVNth\ s\ uid\ PID\ N \wedge$
 $eqExcPID-N\ s\ s1 \wedge B\ vl\ vl1$

definition $\Delta 4 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta 4\ s\ vl\ s1\ vl1 \equiv$
 $\exists\ cid\ uid.$
 $PID \in\in\ paperIDs\ s\ cid \wedge phase\ s\ cid \geq revPH \wedge isREVNth\ s\ uid\ PID\ N \wedge$
 $s = s1 \wedge (\exists\ wl.\ vl = map\ (Pair\ disPH)\ wl \wedge vl1 = map\ (Pair\ disPH)\ wl)$

definition $\Delta e :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **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
 $(\exists\ cid.\ PID \in\in\ paperIDs\ s\ cid \wedge phase\ s\ cid > revPH \wedge fst\ (hd\ vl) = revPH)$
 $)$

lemma *istate- $\Delta 1$* :
assumes $B: B\ vl\ vl1$
shows $\Delta 1\ istate\ vl\ istate\ vl1$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta 1$* : *unwind-cont* $\Delta 1\ \{\Delta 1, \Delta 2, \Delta e\}$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta 2$* : *unwind-cont* $\Delta 2\ \{\Delta 2, \Delta 3, \Delta e\}$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta 3$* : *unwind-cont* $\Delta 3\ \{\Delta 3, \Delta 4, \Delta e\}$

<proof>

lemma *unwind-cont- Δ_4* : *unwind-cont Δ_4 { $\Delta_4, \Delta e$ }*
<proof>

definition *K2exit* **where**

K2exit 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-K2exit*: *invarNT (K2exit cid)*
<proof>

lemma *noVal-K2exit*: *noVal (K2exit cid) v*
<proof>

definition *K3exit* **where**

K3exit cid s \equiv *PID* $\in \in paperIDs$ *s cid* \wedge *phase s cid* $>$ *revPH*

lemma *invarNT-K3exit*: *invarNT (K3exit cid)*
<proof>

lemma *noVal-K3exit*: *noVal (K3exit cid) (revPH, u)*
<proof>

lemma *unwind-exit- Δe* : *unwind-exit Δe*
<proof>

theorem *secure*: *secure*
<proof>

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**
 $T (Trans - - ou s') =$
 $(\exists uid \in UIDs.$
 $isREVNth s' uid PID N$
 \vee
 $(\exists cid. PID \in \in paperIDs s' cid \wedge isPC s' cid uid \wedge pref s' uid PID \neq Conflict$
 $\wedge phase s' cid \geq disPH)$
 $)$

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** $B = B$
 $\langle proof \rangle$

lemma $reachNT\text{-}non\text{-}isRevNth\text{-}isPC\text{-}isChair$:

assumes $reachNT s$ **and** $uid \in UIDs$

shows

$\neg isRevNth s cid uid PID N \wedge$
 $(PID \in \in paperIDs s cid \wedge isPC s cid uid \longrightarrow pref s uid PID = Conflict \vee phase$
 $s cid < disPH) \wedge$
 $(PID \in \in paperIDs s cid \wedge isChair s cid uid \longrightarrow pref s uid PID = Conflict \vee$
 $phase s cid < disPH)$
 $\langle proof \rangle$

lemma $T\text{-}\varphi\text{-}\gamma$:

assumes $1: reachNT s$ **and** $2: step s a = (ou,s') \varphi (Trans s a ou s')$

shows $\neg \gamma (Trans s a ou s')$

$\langle proof \rangle$

lemma $eqExcPID\text{-}N\text{-}step\text{-}out$:

assumes $s's1'$: $eqExcPID\text{-}N s s1$

and $step: step s a = (ou,s')$ **and** $step1: step s1 a = (ou1,s1')$

and $sP: reachNT s$ **and** $s1: reach s1$

and PID : $PID \in \in paperIDs\ s\ cid$
and ph : $phase\ s\ cid = revPH \vee phase\ s\ cid = disPH$
and $UIDs$: $userOfA\ a \in UIDs$
shows $ou = ou1$
 $\langle proof \rangle$

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 : $phase\ s\ cid \geq revPH$
and $UIDs$: $userOfA\ a \in UIDs$
and $decs-exit$: $(reviewsPaper\ (paper\ s\ PID))!N \neq [] \wedge (reviewsPaper\ (paper\ s1\ PID))!N \neq []$
shows $ou = ou1$
 $\langle proof \rangle$

lemma $eqExcPID-N-step-eqExcPID-N2$:
assumes rs : $reach\ s$
and a : $a = Uact\ (uReview\ cid\ uid\ p\ PID\ N\ rc) \vee$
 $a = UUact\ (uuReview\ cid\ uid\ p\ PID\ N\ rc)$ (**is** $?L \vee ?R$)
and $ss1$: $eqExcPID-N\ s\ s1$
and $step$: $step\ s\ a = (outOK, s')$ **and** $step1$: $step\ s1\ a = (outOK, s1')$
shows $eqExcPID-N2\ s'\ s1'$
 $\langle proof \rangle$

lemma $eqExcPID-N-step-\varphi-eqExcPID-N2$:
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 : $\varphi\ (Trans\ s\ a\ ou\ s')$
shows $eqExcPID-N2\ s'\ s1'$
 $\langle proof \rangle$

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**
 $\Delta2\ s\ vl\ s1\ vl1 \equiv$
 $\exists\ cid.$
 $PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid = revPH \wedge \neg (\exists\ uid.\ isREVNth\ s\ uid$
 $PID\ N) \wedge$
 $s = s1 \wedge B\ vl\ vl1$

definition $\Delta3$:: $state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta3\ s\ vl\ s1\ vl1 \equiv$

$\exists cid uid.$
 $PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid \in \{revPH, disPH\} \wedge isREVNth\ s\ uid$
 $PID\ N \wedge$
 $eqExcPID-N\ s\ s1 \wedge B\ vl\ vl1$

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- Δ_1* :

assumes $B: B\ vl\ vl1$

shows $\Delta_1\ istate\ vl\ istate\ vl1$

<proof>

lemma *unwind-cont- Δ_1* : *unwind-cont* $\Delta_1\ \{\Delta_1, \Delta_2, \Delta_e\}$

<proof>

lemma *unwind-cont- Δ_2* : *unwind-cont* $\Delta_2\ \{\Delta_2, \Delta_3, \Delta_e\}$

<proof>

lemma *unwind-cont- Δ_3* : *unwind-cont* $\Delta_3\ \{\Delta_3, \Delta_4, \Delta_e\}$

<proof>

lemma *unwind-cont- Δ_4* : *unwind-cont* $\Delta_4\ \{\Delta_4, \Delta_e\}$

<proof>

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*)

<proof>

lemma *noVal-K1exit*: *noVal (K1exit cid) v*
 ⟨*proof*⟩

definition *K2exit* **where**
K2exit cid s \equiv *PID* $\in\in$ *paperIDs s cid* \wedge *phase s cid* $>$ *disPH*

lemma *invarNT-K2exit*: *invarNT (K2exit cid)*
 ⟨*proof*⟩

lemma *noVal-K2exit*: *noVal (K2exit cid) v*
 ⟨*proof*⟩

lemma *unwind-exit- Δe* : *unwind-exit Δe*
 ⟨*proof*⟩

theorem *secure*: *secure*
 ⟨*proof*⟩

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 $T :: (state, act, out) \text{ trans} \Rightarrow \text{bool}$ **where**
 $T (Trans \text{ - - } ou \ s') =$
 $(\exists \text{ uid} \in \text{UIDs}.$
 $\quad \text{isREVNth } s' \text{ uid } \text{PID } N$
 $\quad \vee$
 $\quad (\exists \text{ cid. } \text{PID} \in \in \text{paperIDs } s' \text{ cid} \wedge \text{isPC } s' \text{ cid } \text{uid} \wedge \text{pref } s' \text{ uid } \text{PID} \neq \text{Conflict}$
 $\wedge \text{phase } s' \text{ cid} \geq \text{disPH})$
 $\quad \vee$
 $\quad (\exists \text{ cid. } \text{PID} \in \in \text{paperIDs } s' \text{ cid} \wedge \text{isPC } s' \text{ cid } \text{uid} \wedge \text{phase } s' \text{ cid} \geq \text{notifPH})$
 $\quad \vee$
 $\quad \text{isAUT } s' \text{ uid } \text{PID} \wedge (\exists \text{ cid. } \text{PID} \in \in \text{paperIDs } s' \text{ cid} \wedge \text{phase } s' \text{ cid} \geq \text{notifPH})$
 $)$

declare $T.simps$ [simp del]

definition $B :: \text{value list} \Rightarrow \text{value list} \Rightarrow \text{bool}$ **where**
 $B \text{ vl } \text{vl1} \equiv \text{vl} \neq []$

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** $B = B$
 $\langle \text{proof} \rangle$

lemma $\text{reachNT-non-isPC-isChair}$:
assumes $\text{reachNT } s$ **and** $\text{uid} \in \text{UIDs}$
shows
 $\neg \text{isRevNth } s \text{ cid } \text{uid } \text{PID } N \wedge$
 $(\text{PID} \in \in \text{paperIDs } s \text{ cid} \wedge \text{isPC } s \text{ cid } \text{uid} \longrightarrow$
 $\quad (\text{pref } s \text{ uid } \text{PID} = \text{Conflict} \vee \text{phase } s \text{ cid} < \text{disPH}) \wedge \text{phase } s \text{ cid} < \text{notifPH})$
 \wedge
 $(\text{PID} \in \in \text{paperIDs } s \text{ cid} \wedge \text{isChair } s \text{ cid } \text{uid} \longrightarrow$
 $\quad (\text{pref } s \text{ uid } \text{PID} = \text{Conflict} \vee \text{phase } s \text{ cid} < \text{disPH}) \wedge \text{phase } s \text{ cid} < \text{notifPH})$
 \wedge
 $(\text{isAut } s \text{ cid } \text{uid } \text{PID} \longrightarrow \text{phase } s \text{ cid} < \text{notifPH})$
 $\langle \text{proof} \rangle$

lemma $T\text{-}\varphi\text{-}\gamma$:
assumes $1: \text{reachNT } s$ **and** $2: \text{step } s \ a = (ou, s')$ $\varphi (Trans \ s \ a \ ou \ s')$
shows $\neg \gamma (Trans \ s \ a \ ou \ s')$
 $\langle \text{proof} \rangle$

lemma $\text{eqExcPID-N-step-out}$:
assumes $s's1'$: $\text{eqExcPID-N } s \ s1$
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 $\text{PID: } \text{PID} \in \in \text{paperIDs } s \ \text{cid}$
and $\text{UIDs: } \text{userOfA } a \in \text{UIDs}$
shows $ou = ou1$
 $\langle \text{proof} \rangle$

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 $\Delta 1\ istate\ vl\ istate\ vl1$

<proof>

lemma *unwind-cont- $\Delta 1$* : *unwind-cont* $\Delta 1\ \{\Delta 1, \Delta 2, \Delta e\}$

<proof>

lemma *unwind-cont- $\Delta 2$* : *unwind-cont* $\Delta 2\ \{\Delta 2, \Delta 3, \Delta e\}$

<proof>

lemma *unwind-cont- $\Delta 3$* : *unwind-cont* $\Delta 3\ \{\Delta 3, \Delta 4, \Delta e\}$

<proof>

lemma *unwind-cont- $\Delta 4$* : *unwind-cont* $\Delta 4\ \{\Delta 4, \Delta e\}$

<proof>

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*)
 <proof>

lemma *noVal-K1exit*: *noVal* (*K1exit cid*) *v*
 <proof>

lemma *unwind-exit-Δe*: *unwind-exit* Δe
 <proof>

theorem *secure*: *secure*
 <proof>

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*)
<proof>

lemma *eqExcD-eq*[*simp,intro!*]: *eqExcD pap pap*
<proof>

lemma *eqExcD-sym*:
assumes *eqExcD pap pap1*
shows *eqExcD pap1 pap*
<proof>

lemma *eqExcD-trans*:
assumes *eqExcD pap pap1* **and** *eqExcD pap1 pap2*
shows *eqExcD pap pap2*
<proof>

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*
<proof>

lemma *eeqExcPID-sym*:
assumes *eeqExcPID s s1* **shows** *eeqExcPID s1 s*
<proof>

lemma *eeqExcPID-trans*:
assumes *eeqExcPID s s1* **and** *eeqExcPID s1 s2* **shows** *eeqExcPID s s2*
<proof>

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*

<proof>

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))*

<proof>

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)

<proof>

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*

<proof>

lemma *eqExcPID-sym*:

assumes *eqExcPID s s1* **shows** *eqExcPID s1 s*

<proof>

lemma *eqExcPID-trans*:

assumes *eqExcPID s s1* **and** *eqExcPID s1 s2* **shows** *eqExcPID s s2*

<proof>

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
 $pref\ s = pref\ s1 \wedge$
 $voronkov\ s = voronkov\ s1 \wedge$
 $news\ s = news\ s1 \wedge phase\ s = phase\ s1 \wedge$

$getAllPaperIDs\ s = getAllPaperIDs\ s1 \wedge$
 $isRev\ s\ cid\ uid\ pid = isRev\ s1\ cid\ uid\ pid \wedge$
 $getReviewIndex\ s\ cid\ uid\ pid = getReviewIndex\ s1\ cid\ uid\ pid \wedge$
 $getRevRole\ s\ cid\ uid\ pid = getRevRole\ s1\ cid\ uid\ pid$
 $\langle proof \rangle$

lemma $eqExcPID-imp1$:
 $eqExcPID\ s\ s1 \implies eqExcD\ (paper\ s\ pid)\ (paper\ s1\ pid)$
 $eqExcPID\ s\ s1 \implies pid \neq PID \vee PID \neq pid \implies$
 $paper\ s\ pid = paper\ s1\ pid \wedge$
 $getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$
 $\langle proof \rangle$

lemma $eqExcPID-imp2$:
assumes $eqExcPID\ s\ s1$ **and** $pid \neq PID \vee PID \neq pid$
shows $getReviewersReviews\ s\ cid\ pid = getReviewersReviews\ s1\ cid\ pid$
 $\langle proof \rangle$

lemma $eqExcPID-RDD$:
 $eqExcPID\ s\ s1 \implies$
 $titlePaper\ (paper\ s\ PID) = titlePaper\ (paper\ s1\ PID) \wedge$
 $abstractPaper\ (paper\ s\ PID) = abstractPaper\ (paper\ s1\ PID) \wedge$
 $contentPaper\ (paper\ s\ PID) = contentPaper\ (paper\ s1\ PID) \wedge$
 $reviewsPaper\ (paper\ s\ PID) = reviewsPaper\ (paper\ s1\ PID) \wedge$
 $decsPaper\ (paper\ s\ PID) = decsPaper\ (paper\ s1\ PID)$
 $\langle proof \rangle$

lemma $eqExcPID-cong[simp, intro]$:
 $\wedge\ uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|confIDs := uu1\!|))$
 $(s1\ (\!|confIDs := uu2\!|))$
 $\wedge\ uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|conf := uu1\!|))\ (s1$
 $(\!|conf := uu2\!|))$

$\wedge\ uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|userIDs := uu1\!|))$
 $(s1\ (\!|userIDs := uu2\!|))$
 $\wedge\ uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|pass := uu1\!|))\ (s1$
 $(\!|pass := uu2\!|))$
 $\wedge\ uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|user := uu1\!|))\ (s1$
 $(\!|user := uu2\!|))$
 $\wedge\ uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|roles := uu1\!|))\ (s1$
 $(\!|roles := uu2\!|))$

$\wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s \langle paperIDs := uu1 \rangle)$
 $(s1 \langle paperIDs := uu2 \rangle)$
 $\wedge uu1 uu2. eqExcPID s s1 \implies eqExcPID uu1 uu2 \implies eqExcPID (s \langle paper := uu1 \rangle)$
 $(s1 \langle paper := uu2 \rangle)$

$\wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s \langle pref := uu1 \rangle)$
 $(s1 \langle pref := uu2 \rangle)$
 $\wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s \langle voronkov := uu1 \rangle)$
 $(s1 \langle voronkov := uu2 \rangle)$
 $\wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s \langle news := uu1 \rangle)$
 $(s1 \langle news := uu2 \rangle)$
 $\wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s \langle phase := uu1 \rangle)$
 $(s1 \langle phase := uu2 \rangle)$
 $\langle proof \rangle$

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 $title = title1 \wedge abstract = abstract1 \wedge content = content1 \wedge reviews = reviews1 \wedge decs = decs1$

$\langle proof \rangle$

7.2 Value Setup

type-synonym $value = string$

fun $\varphi :: (state,act,out) trans \Rightarrow bool$ **where**

$\varphi (Trans - (UUact (uuDis cid uid p pid com)) ou -) = (pid = PID \wedge ou = outOK)$

$\varphi - = False$

lemma φ -def2:

$\varphi (Trans s a ou s') = (\exists cid uid p com. a = UUact (uuDis cid uid p PID com) \wedge ou = outOK)$

$\langle proof \rangle$

fun $f :: (state,act,out) trans \Rightarrow value$ **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 $eqExcPID s s'$

$\langle proof \rangle$

lemma φ -step-eqExcPID:

assumes φ : $\varphi (Trans s a ou s')$

and s : $step\ s\ a = (ou, s')$
shows $eqExcPID\ s\ s'$
 $\langle proof \rangle$

lemma $eqExcPID$ -step:
assumes $s's1'$: $eqExcPID\ s\ s1$
and $step$: $step\ s\ a = (ou, s')$
and $step1$: $step\ s1\ a = (ou1, s1')$
shows $eqExcPID\ s'\ s1'$
 $\langle proof \rangle$

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 φ : $\varphi\ (Trans\ s\ a\ ou\ s')$
shows $\varphi\ (Trans\ s1\ a\ ou1\ s1')$
 $\langle proof \rangle$

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 $\varphi\ (Trans\ s\ a\ ou\ s') = \varphi\ (Trans\ s1\ a\ ou1\ s1')$
 $\langle proof \rangle$

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**
 $T\ (Trans\ -\ -\ ou\ s') =$
 $(\exists\ uid \in UIDs. \exists\ cid.$
 $\quad PID \in \in\ paperIDs\ s'\ cid \wedge isPC\ s'\ cid\ uid \wedge pref\ s'\ uid\ PID \neq Conflict \wedge phase$
 $\quad s'\ cid \geq disPH$
 $\quad)$

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** $B = B$
{*proof*}

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
($PID \in \in paperIDs\ s\ cid \wedge isChair\ s\ cid\ uid \wedge phase\ s\ cid \geq disPH \longrightarrow pref\ s$
 $uid\ PID = Conflict$)
{*proof*}

lemma $T\text{-}\varphi\text{-}\gamma$:
assumes $1: reachNT\ s$ **and** $2: step\ s\ a = (ou, s')$ $\varphi\ (Trans\ s\ a\ ou\ s')$
shows $\neg \gamma\ (Trans\ s\ a\ ou\ s')$
{*proof*}

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 $UIDs: userOfA\ a \in UIDs$
shows $ou = ou1$
{*proof*}

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**
 $\Delta2\ s\ vl\ s1\ vl1 \equiv$
 $\exists\ cid\ uid.$
 $PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid = disPH \wedge$
 $isPC\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict$
 $\wedge eqExcPID\ s\ s1$

definition $\Delta3 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta3\ s\ vl\ s1\ vl1 \equiv$

$\exists cid. PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > disPH \wedge eqExcPID\ s\ s1 \wedge vl1 =$
 \square

definition $\Delta e :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta e\ s\ vl\ s1\ vl1 \equiv$
 $vl \neq [] \wedge$
 $(\exists cid. PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid \geq disPH \wedge$
 $\neg (\exists uid. isPC\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict))$
 $)$

lemma *init- $\Delta 1$* :
assumes $B: B\ vl\ vl1$
shows $\Delta 1\ istate\ vl\ istate\ vl1$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta 1$* : *unwind-cont* $\Delta 1\ \{\Delta 1, \Delta 2, \Delta e\}$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta 2$* : *unwind-cont* $\Delta 2\ \{\Delta 2, \Delta 3, \Delta e\}$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta 3$* : *unwind-cont* $\Delta 3\ \{\Delta 3, \Delta e\}$
 $\langle proof \rangle$

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* $(K1exit\ cid)$
 $\langle proof \rangle$

lemma *noVal-K1exit*: *noVal* $(K1exit\ cid)\ v$
 $\langle proof \rangle$

lemma *unwind-exit- Δe* : *unwind-exit* Δe
 $\langle proof \rangle$

theorem *secure*: *secure*
 $\langle proof \rangle$

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 ⇒ paper ⇒ bool where
eqExcD (Paper title abstract ct reviews dis decs)
  (Paper title1 abstract1 ct1 reviews1 dis1 decs1) =
  (title = title1 ∧ abstract = abstract1 ∧ ct = ct1 ∧ reviews = reviews1 ∧ 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*)
(*proof*)

lemma *eqExcD-eq[simp,intro!]*: *eqExcD pap pap*
(*proof*)

lemma *eqExcD-sym*:

assumes *eqExcD pap pap1*
shows *eqExcD pap1 pap*
(*proof*)

lemma *eqExcD-trans*:

assumes *eqExcD pap pap1* **and** *eqExcD pap1 pap2*
shows *eqExcD pap pap2*
(*proof*)

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*
(*proof*)

lemma *eeqExcPID-sym*:

assumes *eeqExcPID s s1* **shows** *eeqExcPID s1 s*
(*proof*)

lemma *eeqExcPID-trans*:

assumes *eeqExcPID s s1* **and** *eeqExcPID s1 s2* **shows** *eeqExcPID s s2*
(*proof*)

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*
(*proof*)

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))*
(*proof*)

lemma *eeqExcPID-RDD*:

$eeqExcPID\ paps\ paps1 \implies$
 $titlePaper\ (paps\ PID) = titlePaper\ (paps1\ PID) \wedge$
 $abstractPaper\ (paps\ PID) = abstractPaper\ (paps1\ PID) \wedge$
 $contentPaper\ (paps\ PID) = contentPaper\ (paps1\ PID) \wedge$
 $reviewsPaper\ (paps\ PID) = reviewsPaper\ (paps1\ PID) \wedge$
 $disPaper\ (paps\ PID) = disPaper\ (paps1\ PID)$
 <proof>

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 $eqExcPID-eq[simp,intro!]: eqExcPID\ s\ s$
 <proof>

lemma $eqExcPID-sym:$
assumes $eqExcPID\ s\ s1$ **shows** $eqExcPID\ s1\ s$
 <proof>

lemma $eqExcPID-trans:$
assumes $eqExcPID\ s\ s1$ **and** $eqExcPID\ s1\ s2$ **shows** $eqExcPID\ s\ s2$
 <proof>

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
 $pref\ s = pref\ s1 \wedge$
 $voronkov\ s = voronkov\ s1 \wedge$
 $news\ s = news\ s1 \wedge phase\ s = phase\ s1 \wedge$

 $getAllPaperIDs\ s = getAllPaperIDs\ s1 \wedge$
 $isRev\ s\ cid\ uid\ pid = isRev\ s1\ cid\ uid\ pid \wedge$

$getReviewIndex\ s\ cid\ uid\ pid = getReviewIndex\ s1\ cid\ uid\ pid \wedge$
 $getRevRole\ s\ cid\ uid\ pid = getRevRole\ s1\ cid\ uid\ pid$
 <proof>

lemma *eqExcPID-imp1*:

$eqExcPID\ s\ s1 \implies eqExcD\ (paper\ s\ pid)\ (paper\ s1\ pid)$
 $eqExcPID\ s\ s1 \implies pid \neq PID \vee PID \neq pid \implies$
 $paper\ s\ pid = paper\ s1\ pid \wedge$
 $getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$
 <proof>

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>

lemma *eqExcPID-RDD*:

$eqExcPID\ s\ s1 \implies$
 $titlePaper\ (paper\ s\ PID) = titlePaper\ (paper\ s1\ PID) \wedge$
 $abstractPaper\ (paper\ s\ PID) = abstractPaper\ (paper\ s1\ PID) \wedge$
 $contentPaper\ (paper\ s\ PID) = contentPaper\ (paper\ s1\ PID) \wedge$
 $reviewsPaper\ (paper\ s\ PID) = reviewsPaper\ (paper\ s1\ PID) \wedge$
 $disPaper\ (paper\ s\ PID) = disPaper\ (paper\ s1\ PID)$
 <proof>

lemma *eqExcPID-cong[simp, intro]*:

$\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|confIDs := uu1|))$
 $(s1\ (\!|confIDs := uu2|))$
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|conf := uu1|))\ (s1$
 $(\!|conf := uu2|))$

$\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|userIDs := uu1|))$
 $(s1\ (\!|userIDs := uu2|))$

$\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|pass := uu1|))\ (s1$
 $(\!|pass := uu2|))$

$\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|user := uu1|))\ (s1$
 $(\!|user := uu2|))$

$\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|roles := uu1|))\ (s1$
 $(\!|roles := uu2|))$

$\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|paperIDs := uu1|))$
 $(s1\ (\!|paperIDs := uu2|))$

$\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies eqExcPID\ uu1\ uu2 \implies eqExcPID\ (s\ (\!|paper :=$
 $uu1|))\ (s1\ (\!|paper := uu2|))$

$\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|pref := uu1|))\ (s1$
 $(\!|pref := uu2|))$

$\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|voronkov := uu1|))$
 $(s1\ (\!|voronkov := uu2|))$

$\wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s \langle news := uu1 \rangle) (s1 \langle news := uu2 \rangle)$
 $\wedge uu1 uu2. eqExcPID s s1 \implies uu1 = uu2 \implies eqExcPID (s \langle phase := uu1 \rangle) (s1 \langle phase := uu2 \rangle)$
 $\langle proof \rangle$

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 $title = title1 \wedge abstract = abstract1 \wedge content = content1 \wedge reviews = reviews1 \wedge dis = dis1$
 $\langle proof \rangle$

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* :: $paper \Rightarrow paper \Rightarrow bool$ **where**
 $eqExcD2 (Paper title abstract ct reviews dis decs)$
 $(Paper title1 abstract1 ct1 reviews1 dis1 decs1) =$
 $(title = title1 \wedge abstract = abstract1 \wedge ct = ct1 \wedge reviews = reviews1 \wedge dis = dis1 \wedge$
 $hd decs = hd decs1)$

lemma *eqExcD2*:
 $eqExcD2 pap pap1 =$
 $(titlePaper pap = titlePaper pap1 \wedge abstractPaper pap = abstractPaper pap1 \wedge$
 $contentPaper pap = contentPaper pap1 \wedge$
 $reviewsPaper pap = reviewsPaper pap1 \wedge disPaper pap = disPaper pap1 \wedge$
 $hd (decsPaper pap) = hd (decsPaper pap1)$
 $)$
 $\langle proof \rangle$

lemma *eqExcD2-eq[simp,intro!]*: $eqExcD2 pap pap$
 $\langle proof \rangle$

lemma *eqExcD2-sym*:
assumes $eqExcD2 pap pap1$
shows $eqExcD2 pap1 pap$
 $\langle proof \rangle$

lemma *eqExcD2-trans*:
assumes $eqExcD2 pap pap1$ **and** $eqExcD2 pap1 pap2$
shows $eqExcD2 pap pap2$
 $\langle proof \rangle$

definition *eeqExcPID2* **where**
 $eeqExcPID2 paps paps1 \equiv$

$\forall pid.$ if $pid = PID$ then $eqExcD2 (paps pid) (paps1 pid)$ else $paps pid = paps1 pid$

lemma $eeqExcPID2$ - eeq [*simp,intro!*]: $eeqExcPID2 s s$
 $\langle proof \rangle$

lemma $eeqExcPID2$ -*sym*:
assumes $eeqExcPID2 s s1$ **shows** $eeqExcPID2 s1 s$
 $\langle proof \rangle$

lemma $eeqExcPID2$ -*trans*:
assumes $eeqExcPID2 s s1$ **and** $eeqExcPID2 s1 s2$ **shows** $eeqExcPID2 s s2$
 $\langle proof \rangle$

lemma $eeqExcPID2$ -*imp*:
 $eeqExcPID2 paps paps1 \implies eqExcD2 (paps PID) (paps1 PID)$
 $\llbracket eeqExcPID2 paps paps1; pid \neq PID \rrbracket \implies paps pid = paps1 pid$
 $\langle proof \rangle$

lemma $eeqExcPID2$ -*cong*:
assumes $eeqExcPID2 paps paps1$
and $pid = PID \implies eqExcD2 uu uu1$
and $pid \neq PID \implies uu = uu1$
shows $eeqExcPID2 (paps (pid := uu)) (paps1 (pid := uu1))$
 $\langle proof \rangle$

lemma $eeqExcPID2$ -*RDD*:
 $eeqExcPID2 paps paps1 \implies$
 $titlePaper (paps PID) = titlePaper (paps1 PID) \wedge$
 $abstractPaper (paps PID) = abstractPaper (paps1 PID) \wedge$
 $contentPaper (paps PID) = contentPaper (paps1 PID) \wedge$
 $reviewsPaper (paps PID) = reviewsPaper (paps1 PID) \wedge$
 $disPaper (paps PID) = disPaper (paps1 PID) \wedge$
 $hd (decsPaper (paps PID)) = hd (decsPaper (paps1 PID))$
 $\langle proof \rangle$

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 $eqExcPID2-eq[simp,intro!]$: $eqExcPID2\ s\ s$
(proof)

lemma $eqExcPID2-sym$:
assumes $eqExcPID2\ s\ s1$ **shows** $eqExcPID2\ s1\ s$
(proof)

lemma $eqExcPID2-trans$:
assumes $eqExcPID2\ s\ s1$ **and** $eqExcPID2\ s1\ s2$ **shows** $eqExcPID2\ s\ s2$
(proof)

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
 $pref\ s = pref\ s1 \wedge$
 $voronkov\ s = voronkov\ s1 \wedge$
 $news\ s = news\ s1 \wedge phase\ s = phase\ s1 \wedge$

 $getAllPaperIDs\ s = getAllPaperIDs\ s1 \wedge$
 $isRev\ s\ cid\ uid\ pid = isRev\ s1\ cid\ uid\ pid \wedge$
 $getReviewIndex\ s\ cid\ uid\ pid = getReviewIndex\ s1\ cid\ uid\ pid \wedge$
 $getRevRole\ s\ cid\ uid\ pid = getRevRole\ s1\ cid\ uid\ pid$
(proof)

lemma $eqExcPID2-imp1$:
 $eqExcPID2\ s\ s1 \implies eqExcD2\ (paper\ s\ pid)\ (paper\ s1\ pid)$
 $eqExcPID2\ s\ s1 \implies pid \neq PID \vee PID \neq pid \implies$
 $paper\ s\ pid = paper\ s1\ pid \wedge$
 $getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$
(proof)

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)

lemma $eqExcPID2-RDD$:
 $eqExcPID2\ s\ s1 \implies$
 $titlePaper\ (paper\ s\ PID) = titlePaper\ (paper\ s1\ PID) \wedge$
 $abstractPaper\ (paper\ s\ PID) = abstractPaper\ (paper\ s1\ PID) \wedge$
 $contentPaper\ (paper\ s\ PID) = contentPaper\ (paper\ s1\ PID) \wedge$

$reviewsPaper (paper\ s\ PID) = reviewsPaper (paper\ s1\ PID) \wedge$
 $disPaper (paper\ s\ PID) = disPaper (paper\ s1\ PID)$
 <proof>

lemma *eqExcPID2-cong[simp, intro]:*

$\wedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|confIDs := uu1|))$
 $(s1\ (\!|confIDs := uu2|))$

$\wedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|conf := uu1|))$
 $(s1\ (\!|conf := uu2|))$

$\wedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|userIDs := uu1|))$
 $(s1\ (\!|userIDs := uu2|))$

$\wedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|pass := uu1|))$
 $(s1\ (\!|pass := uu2|))$

$\wedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|user := uu1|))$
 $(s1\ (\!|user := uu2|))$

$\wedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|roles := uu1|))$
 $(s1\ (\!|roles := uu2|))$

$\wedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|paperIDs := uu1|))$
 $(s1\ (\!|paperIDs := uu2|))$

$\wedge uu1\ uu2. eqExcPID2\ s\ s1 \implies eqExcPID2\ uu1\ uu2 \implies eqExcPID2\ (s\ (\!|paper := uu1|))$
 $(s1\ (\!|paper := uu2|))$

$\wedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|pref := uu1|))$
 $(s1\ (\!|pref := uu2|))$

$\wedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|voronkov := uu1|))$
 $(s1\ (\!|voronkov := uu2|))$

$\wedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|news := uu1|))$
 $(s1\ (\!|news := uu2|))$

$\wedge uu1\ uu2. eqExcPID2\ s\ s1 \implies uu1 = uu2 \implies eqExcPID2\ (s\ (\!|phase := uu1|))$
 $(s1\ (\!|phase := uu2|))$

<proof>

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 $title = title1 \wedge abstract = abstract1 \wedge content = content1 \wedge reviews = reviews1 \wedge$

$dis = dis1$

<proof>

8.2 Value Setup

type-synonym $value = decision$

fun $\varphi :: (state, act, out)\ trans \Rightarrow bool$ **where**

$\varphi (Trans - (UAct (uuDec\ cid\ uid\ p\ pid\ dec))\ ou\ -) = (pid = PID \wedge ou = outOK)$

|
 $\varphi - = \text{False}$

lemma $\varphi\text{-def2}$:

$\varphi (\text{Trans } s \ a \ ou \ s') = (\exists \ cid \ uid \ p \ dec. \ a = \text{UUact } (uuDec \ cid \ uid \ p \ PID \ dec) \wedge$
 $ou = \text{outOK})$
 $\langle \text{proof} \rangle$

fun $f :: (\text{state}, \text{act}, \text{out}) \text{ trans} \Rightarrow \text{value}$ **where**
 $f (\text{Trans } - \ (\text{UUact } (uuDec \ cid \ uid \ p \ pid \ dec)) \ -) = dec$

lemma $\text{UUact-uuDec-step-eqExcPID}$:

assumes $a: a = \text{UUact } (uuDec \ cid \ uid \ p \ PID \ dec)$
and $\text{step } s \ a = (ou, s')$
shows $\text{eqExcPID } s \ s'$
 $\langle \text{proof} \rangle$

lemma $\varphi\text{-step-eqExcPID}$:

assumes $\varphi: \varphi (\text{Trans } s \ a \ ou \ s')$
and $s: \text{step } s \ a = (ou, s')$
shows $\text{eqExcPID } s \ s'$
 $\langle \text{proof} \rangle$

lemma eqExcPID-step :

assumes $s's1': \text{eqExcPID } s \ s1'$
and $\text{step}: \text{step } s \ a = (ou, s')$
and $\text{step1}: \text{step } s1 \ a = (ou1, s1')$
shows $\text{eqExcPID } s' \ s1'$
 $\langle \text{proof} \rangle$

lemma $\text{eqExcPID-step-}\varphi\text{-imp}$:

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 $\varphi: \varphi (\text{Trans } s \ a \ ou \ s')$
shows $\varphi (\text{Trans } s1 \ a \ ou1 \ s1')$
 $\langle \text{proof} \rangle$

lemma $\text{eqExcPID-step-}\varphi$:

assumes $s's1': \text{eqExcPID } s \ s1'$
and $\text{step}: \text{step } s \ a = (ou, s')$ **and** $\text{step1}: \text{step } s1 \ a = (ou1, s1')$
shows $\varphi (\text{Trans } s \ a \ ou \ s') = \varphi (\text{Trans } s1 \ a \ ou1 \ s1')$
 $\langle \text{proof} \rangle$

lemma eqExcPID2-step :

assumes $s's1': \text{eqExcPID2 } s \ s1'$
and $\text{step}: \text{step } s \ a = (ou, s')$

and *step1*: *step s1 a = (ou1,s1')*
shows *eqExcPID2 s' s1'*
<proof>

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')*
<proof>

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')*
<proof>

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 ⇒ value list ⇒ bool where*
B vl vl1 ≡ vl ≠ [] ∧ vl1 ≠ [] ∧ last vl = last vl1

interpretation *BD-Security-IO where*
istate = istate and step = step and
φ = φ and f = f and γ = γ and g = g and T = T and B = B

$\langle proof \rangle$

lemma *reachNT-non-isPC-isChair*:

assumes *reachNT s* **and** *uid* \in *UIDs*

shows

$(PID \in \in paperIDs\ s\ cid \wedge isPC\ s\ cid\ uid \wedge phase\ s\ cid \geq disPH \longrightarrow pref\ s\ uid$
 $PID = Conflict) \wedge$

$(PID \in \in paperIDs\ s\ cid \wedge isChair\ s\ cid\ uid \wedge phase\ s\ cid \geq disPH \longrightarrow pref\ s$
 $uid\ PID = Conflict)$

$\langle proof \rangle$

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'*)

$\langle proof \rangle$

lemma *eqExcPID2-eqExcPID*:

eqExcPID2 s s1 \implies *eqExcPID s s1*

$\langle proof \rangle$

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 *ph*: *phase s cid = disPH*

and *UIDs*: *userOfA a* \in *UIDs*

shows *ou = ou1*

$\langle proof \rangle$

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* $\in \in$ *paperIDs s cid*

and *ph*: *phase s cid* \geq *disPH*

and *UIDs*: *userOfA a* \in *UIDs*

and *decs-exit*: *decsPaper (paper s PID)* \neq \square *decsPaper (paper s1 PID)* \neq \square

shows *ou = ou1*

$\langle proof \rangle$

lemma *eqExcPID-step-eqExcPID2*:

assumes *a*: *a = UUact (uuDec cid uid p PID dec)*

and *ss1*: *eqExcPID s s1*

and *step*: *step s a = (outOK,s')* **and** *step1*: *step s1 a = (outOK,s1')*

and *s*: *reach s reach s1* **and** *PID*: *PID* $\in \in$ *paperIDs s cid* **and** *ph*: *phase s cid* $<$
notifPH

shows *eqExcPID2 s' s1'*

$\langle proof \rangle$

lemma *eqExcPID-step-φ-eqExcPID2*:
assumes *ss1*: *eqExcPID s s1*
and *step*: *step s a = (ou,s')* **and** *step1*: *step s1 a = (ou1,s1')*
and *φ*: *φ (Trans s a ou s')*
and *s*: *reach s reach s1* **and** *PID*: *PID ∈∈ paperIDs s cid* **and** *ph*: *phase s cid ≤ disPH*
shows *eqExcPID2 s' s1'*
<proof>

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**
 $\Delta 2\ s\ vl\ s1\ vl1 \equiv$
 $\exists\ cid\ uid.$
 $PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid = disPH \wedge$
 $isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict \wedge$
 $eqExcPID\ s\ s1 \wedge B\ vl\ vl1$

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-Δ1*:
assumes *B*: *B vl vl1*
shows $\Delta 1\ istate\ vl\ istate\ vl1$
<proof>

lemma *unwind-cont-Δ1*: *unwind-cont Δ1 {Δ1,Δ2,Δe}*
<proof>

lemma *unwind-cont- $\Delta 2$* : *unwind-cont* $\Delta 2$ $\{\Delta 2, \Delta 3, \Delta e\}$
<proof>

lemma *unwind-cont- $\Delta 3$* : *unwind-cont* $\Delta 3$ $\{\Delta 3, \Delta e\}$
<proof>

definition *K1exit* **where**

K1exit cid s \equiv
phase s cid \geq *disPH* \wedge *PID* \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*)
<proof>

lemma *noVal-K1exit*: *noVal* (*K1exit cid*) *v*
<proof>

definition *K2exit* **where**

K2exit cid s \equiv *PID* \in *paperIDs s cid* \wedge *phase s cid* $>$ *disPH*

lemma *invarNT-K2exit*: *invarNT* (*K2exit cid*)
<proof>

lemma *noVal-K2exit*: *noVal* (*K2exit cid*) *v*
<proof>

lemma *unwind-exit- Δe* : *unwind-exit* Δe
<proof>

theorem *secure*: *secure*
<proof>

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** $B = B$
 $\langle proof \rangle$

lemma $reachNT\text{-}non\text{-}isPC\text{-}isChair$:
assumes $reachNT s$ **and** $uid \in UIDs$

shows

$(PID \in \in paperIDs s cid \wedge isPC s cid uid \wedge phase s cid \geq disPH$
 $\longrightarrow pref s uid PID = Conflict \wedge phase s cid < notifPH) \wedge$
 $(PID \in \in paperIDs s cid \wedge isChair s cid uid \wedge phase s cid \geq disPH$
 $\longrightarrow pref s uid PID = Conflict \wedge phase s cid < notifPH) \wedge$
 $(isAut s cid uid PID \longrightarrow phase s cid < notifPH)$
 $\langle proof \rangle$

lemma $T\text{-}\varphi\text{-}\gamma$:

assumes $1: reachNT s$ **and** $2: step s a = (ou, s') \varphi (Trans s a ou s')$

shows $\neg \gamma (Trans s a ou s')$

$\langle proof \rangle$

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 *UIDs*: $userOfA\ a \in UIDs$
shows $ou = ou1$
 $\langle proof \rangle$

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**
 $\Delta2\ s\ vl\ s1\ vl1 \equiv$
 $\exists\ cid\ uid.$
 $PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid = disPH \wedge$
 $isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict \wedge$
 $eqExcPID\ s\ s1$

definition $\Delta3 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta3\ s\ vl\ s1\ vl1 \equiv$
 $\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > disPH \wedge eqExcPID\ s\ s1 \wedge vl1 =$
 \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- $\Delta1$* :
assumes $B: B\ vl\ vl1$
shows $\Delta1\ istate\ vl\ istate\ vl1$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta1$* : $unwind-cont\ \Delta1\ \{\Delta1, \Delta2, \Delta e\}$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta2$* : $unwind-cont\ \Delta2\ \{\Delta2, \Delta3, \Delta e\}$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta3$* : $unwind-cont\ \Delta3\ \{\Delta3, \Delta e\}$
 $\langle proof \rangle$

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)*
<proof>

lemma *noVal-K1exit: noVal (K1exit cid) v*
<proof>

definition *K2exit where*
K2exit cid s \equiv PID \in paperIDs s cid \wedge phase s cid $>$ disPH

lemma *invarNT-K2exit: invarNT (K2exit cid)*
<proof>

lemma *noVal-K2exit: noVal (K2exit cid) v*
<proof>

lemma *unwind-exit- Δe : unwind-exit Δe*
<proof>

theorem *secure: secure*
<proof>

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 values³ is needed in order to express the piece of information (knowledge) that the appointed reviewers are among the non-conflicted PC members.⁴

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)
- 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 $r \in \in rl \longleftrightarrow r \in \in rl1$

<proof>

lemmas *eqExcRLR* = *eqExcRLR-def*

³This is because conflicts can no longer be changed at the time when reviewers can be appointed, i.e., in the reviewing phase.

⁴In CoCon, only PC members can be appointed as reviewers; there is no subreviewing facility.

lemma *eqExcRLR-eq[simp,intro!]*: *eqExcRLR rl rl*
(*proof*)

lemma *eqExcRLR-sym*:
assumes *eqExcRLR rl rl1*
shows *eqExcRLR rl1 rl*
(*proof*)

lemma *eqExcRLR-trans*:
assumes *eqExcRLR rl rl1* **and** *eqExcRLR rl1 rl2*
shows *eqExcRLR rl rl2*
(*proof*)

lemma *eqExcRLR-imp*:
assumes *s: reach s* **and** *pid: pid ≠ PID* **and**
1: eqExcRLR (roles s cid uid) (roles s1 cid uid)
shows
isRevNth s cid uid pid = isRevNth s1 cid uid pid \wedge
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 (**is** *?A* \wedge *?B* \wedge *?C*
 \wedge *?D*)
(*proof*)

lemma *eqExcRLR-imp2*:
assumes *eqExcRLR (roles s cid uid) (roles s1 cid uid)*
shows
isPC s cid uid = isPC s1 cid uid \wedge
isChair s cid uid = isChair s1 cid uid \wedge
isAut s cid uid = isAut s1 cid uid
(*proof*)

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*
(*proof*)

lemma *arg-cong3*: $a = a1 \implies b = b1 \implies c = c1 \implies h a b c = h a1 b1 c1$
(*proof*)

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*: $a = a1 \implies (a \neq b) = (a1 \neq b)$ (*proof*)

lemma *isRev-pref-notConflict*:
assumes *reach s* **and** *isRev s cid uid pid*
shows *pref s uid pid* \neq *Conflict*
 \langle *proof* \rangle

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*
 \langle *proof* \rangle

lemma *eqExcRLR-imp-isRevRole-imp*:
assumes *eqExcRLR rl rl1*
shows $[r \leftarrow rl. \neg \text{isRevRole } r] = [r \leftarrow rl1 . \neg \text{isRevRole } r]$
 \langle *proof* \rangle

lemma *notIsPC-eqExLRL-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*
and *eq: eqExcRLR (roles s cid uid) (roles (s1::state) cid uid)*
shows *roles s cid uid* = *roles s1 cid uid*
 \langle *proof* \rangle

lemma *foo1*: $P a \implies [r \leftarrow \text{List.insert } a \ l . P \ r] = (\text{if } a \in \text{set } l \text{ then filter } P \ l \text{ else } a \# \text{filter } P \ l)$
 \langle *proof* \rangle

lemma *foo2*: $\llbracket \text{eqExcRLR } rl \ rl'; \neg \text{isRevRoleFor } PID \ x \rrbracket \implies \text{eqExcRLR } (\text{List.insert } x \ rl) \ (\text{List.insert } x \ rl')$
 \langle *proof* \rangle

lemma *foo3*:
assumes *eqExcRLR rl rl' isRevRoleFor PID x*
shows *eqExcRLR (List.insert x rl) (rl')*
and *eqExcRLR (rl) (List.insert x rl')*
 \langle *proof* \rangle

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
 $(\forall \text{ cid uid. eqExcRLR (roles s cid uid) (roles s1 cid uid))$
 \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-eq[simp,intro!]*: *eqExcPID s s*
 <proof>

lemma *eqExcPID-sym*:
assumes *eqExcPID s s1* **shows** *eqExcPID s1 s*
 <proof>

lemma *eqExcPID-trans*:
assumes *eqExcPID s s1* **and** *eqExcPID s1 s2* **shows** *eqExcPID s s2*
 <proof>

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
eqExcRLR (roles s cid uid) (roles s1 cid uid)
 \wedge
 $paperIDs\ s = paperIDs\ s1$
 \wedge
 $paper\ s = paper\ s1$
 \wedge
 $pref\ s = pref\ s1 \wedge$
 $voronkov\ s = voronkov\ s1 \wedge$
 $news\ s = news\ s1 \wedge phase\ s = phase\ s1 \wedge$
 $getAllPaperIDs\ s = getAllPaperIDs\ s1$
 <proof>

lemma *eqExcPID-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>

lemma *eqExcPID-imp1*:
eqExcPID s s1 $\implies pid \neq PID \vee PID \neq pid \implies$
 $getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$
 <proof>

lemma *eqExcPID-imp2*:
assumes *reach s* **and** *eqExcPID s s1* **and** *pid \neq PID \vee PID \neq pid*

shows $getReviewersReviews\ s\ cid\ pid = getReviewersReviews\ s1\ cid\ pid$
 $\langle proof \rangle$

lemma *eqExcPID-imp3*:

$reach\ s \implies eqExcPID\ s\ s1 \implies pid \neq PID \vee PID \neq pid$
 \implies
 $getNthReview\ s\ pid = getNthReview\ s1\ pid$
 $\langle proof \rangle$

lemma *eqExcPID-cong[simp, intro]*:

$\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|confIDs := uu1|))$
 $(s1\ (\!|confIDs := uu2|))$
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|conf := uu1|))\ (s1$
 $(\!|conf := uu2|))$

$\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|userIDs := uu1|))$
 $(s1\ (\!|userIDs := uu2|))$
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|pass := uu1|))\ (s1$
 $(\!|pass := uu2|))$
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|user := uu1|))\ (s1$
 $(\!|user := uu2|))$
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|roles := uu1|))\ (s1$
 $(\!|roles := uu2|))$

$\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|paperIDs := uu1|))$
 $(s1\ (\!|paperIDs := uu2|))$
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|paper := uu1|))\ (s1$
 $(\!|paper := uu2|))$

$\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|pref := uu1|))\ (s1$
 $(\!|pref := uu2|))$
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|voronkov := uu1|))$
 $(s1\ (\!|voronkov := uu2|))$
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|news := uu1|))\ (s1$
 $(\!|news := uu2|))$
 $\bigwedge uu1\ uu2. eqExcPID\ s\ s1 \implies uu1 = uu2 \implies eqExcPID\ (s\ (\!|phase := uu1|))\ (s1$
 $(\!|phase := uu2|))$

$\langle proof \rangle$

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*)
<*proof*>

lemma *eqExcR-eq[simp,intro!]*: *eqExcR pap pap*
<*proof*>

lemma *eqExcR-sym*:

assumes *eqExcR pap pap1*
shows *eqExcR pap1 pap*
<*proof*>

lemma *eqExcR-trans*:

assumes *eqExcR pap pap1* **and** *eqExcR pap1 pap2*
shows *eqExcR pap pap2*
<*proof*>

definition *eeqExcPID* **where**

eeqExcPID paps paps1 \equiv
 \forall *pid*. if *pid* = *PID* then *eqExcR (paps pid) (paps1 pid)* else *paps pid* = *paps1 pid*

lemma *eeqExcPID-eeq[simp,intro!]*: *eeqExcPID s s*
<*proof*>

lemma *eeqExcPID-sym*:

assumes *eeqExcPID s s1* **shows** *eeqExcPID s1 s*
<*proof*>

lemma *eeqExcPID-trans*:

assumes *eeqExcPID s s1* **and** *eeqExcPID s1 s2* **shows** *eeqExcPID s s2*
<*proof*>

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*
<*proof*>*

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))*
<*proof*>

lemma *eeqExcPID-RDD*:

$eeqExcPID\ paps\ paps1 \implies$
 $titlePaper\ (paps\ PID) = titlePaper\ (paps1\ PID) \wedge$
 $abstractPaper\ (paps\ PID) = abstractPaper\ (paps1\ PID) \wedge$
 $contentPaper\ (paps\ PID) = contentPaper\ (paps1\ PID) \wedge$
 $disPaper\ (paps\ PID) = disPaper\ (paps1\ PID) \wedge$
 $decsPaper\ (paps\ PID) = decsPaper\ (paps1\ PID)$
 <proof>

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
 $(\forall\ cid\ uid.\ eqExcRLR\ (roles\ s\ cid\ uid)\ (roles\ s1\ cid\ uid))$
 \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 $eqExcPID2\text{-}eq[simp,intro!]: eqExcPID2\ s\ s$
 <proof>

lemma $eqExcPID2\text{-}sym:$
assumes $eqExcPID2\ s\ s1$ **shows** $eqExcPID2\ s1\ s$
 <proof>

lemma $eqExcPID2\text{-}trans:$
assumes $eqExcPID2\ s\ s1$ **and** $eqExcPID2\ s1\ s2$ **shows** $eqExcPID2\ s\ s2$
 <proof>

lemma $eqExcPID2\text{-}imp:$
 $eqExcPID2\ s\ s1 \implies$
 $confIDs\ s = confIDs\ s1 \wedge conf\ s = conf\ s1 \wedge$
 $userIDs\ s = userIDs\ s1 \wedge pass\ s = pass\ s1 \wedge user\ s = user\ s1$
 \wedge
 $eqExcRLR\ (roles\ s\ cid\ uid)\ (roles\ s1\ cid\ uid)$
 \wedge
 $paperIDs\ s = paperIDs\ s1$
 \wedge
 $eeqExcPID\ (paper\ s)\ (paper\ s1)$
 \wedge
 $pref\ s = pref\ s1 \wedge$
 $voronkov\ s = voronkov\ s1 \wedge$

$news\ s = news\ s1 \wedge phase\ s = phase\ s1 \wedge$

$getAllPaperIDs\ s = getAllPaperIDs\ s1$
{proof}

lemma *eeqExcPID-imp2*:

assumes $pid: pid \neq PID$ **and**

$1: eeqExcPID\ (paper\ s)\ (paper\ s1)$

shows

$reviewsPaper\ (paper\ s\ pid) = reviewsPaper\ (paper\ s1\ pid)$

{proof}

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}

lemma *eqExcPID2-imp1*:

$eqExcPID2\ s\ s1 \implies eqExcR\ (paper\ s\ pid)\ (paper\ s1\ pid)$

$eqExcPID2\ s\ s1 \implies pid \neq PID \vee PID \neq pid \implies$

$paper\ s\ pid = paper\ s1\ pid \wedge$

$getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$

{proof}

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}

lemma *eqExcPID2-imp3*:

$reach\ s \implies eqExcPID2\ s\ s1 \implies pid \neq PID \vee PID \neq pid$

\implies

$getNthReview\ s\ pid = getNthReview\ s1\ pid$

{proof}

lemma *eqExcPID2-RDD*:

$eqExcPID2\ s\ s1 \implies$

$titlePaper\ (paper\ s\ PID) = titlePaper\ (paper\ s1\ PID) \wedge$

$abstractPaper\ (paper\ s\ PID) = abstractPaper\ (paper\ s1\ PID) \wedge$

$contentPaper\ (paper\ s\ PID) = contentPaper\ (paper\ s1\ PID) \wedge$

$disPaper\ (paper\ s\ PID) = disPaper\ (paper\ s1\ PID) \wedge$

$decsPaper\ (paper\ s\ PID) = decsPaper\ (paper\ s1\ PID)$

{proof}

lemma *eqExcPID2-cong[simp, intro]*:

$\wedge uu1 uu2. eqExcPID2 s s1 \implies uu1 = uu2 \implies eqExcPID2 (s (\!|confIDs := uu1\!|))$
 $(s1 (\!|confIDs := uu2\!|))$
 $\wedge uu1 uu2. eqExcPID2 s s1 \implies uu1 = uu2 \implies eqExcPID2 (s (\!|conf := uu1\!|))$
 $(s1 (\!|conf := uu2\!|))$

$\wedge uu1 uu2. eqExcPID2 s s1 \implies uu1 = uu2 \implies eqExcPID2 (s (\!|userIDs := uu1\!|))$
 $(s1 (\!|userIDs := uu2\!|))$
 $\wedge uu1 uu2. eqExcPID2 s s1 \implies uu1 = uu2 \implies eqExcPID2 (s (\!|pass := uu1\!|))$
 $(s1 (\!|pass := uu2\!|))$
 $\wedge uu1 uu2. eqExcPID2 s s1 \implies uu1 = uu2 \implies eqExcPID2 (s (\!|user := uu1\!|))$
 $(s1 (\!|user := uu2\!|))$
 $\wedge uu1 uu2. eqExcPID2 s s1 \implies uu1 = uu2 \implies eqExcPID2 (s (\!|roles := uu1\!|))$
 $(s1 (\!|roles := uu2\!|))$

$\wedge uu1 uu2. eqExcPID2 s s1 \implies uu1 = uu2 \implies eqExcPID2 (s (\!|paperIDs := uu1\!|))$
 $(s1 (\!|paperIDs := uu2\!|))$
 $\wedge uu1 uu2. eqExcPID2 s s1 \implies eqExcPID uu1 uu2 \implies eqExcPID2 (s (\!|paper := uu1\!|))$
 $(s1 (\!|paper := uu2\!|))$

$\wedge uu1 uu2. eqExcPID2 s s1 \implies uu1 = uu2 \implies eqExcPID2 (s (\!|pref := uu1\!|))$
 $(s1 (\!|pref := uu2\!|))$
 $\wedge uu1 uu2. eqExcPID2 s s1 \implies uu1 = uu2 \implies eqExcPID2 (s (\!|voronkov := uu1\!|))$
 $(s1 (\!|voronkov := uu2\!|))$
 $\wedge uu1 uu2. eqExcPID2 s s1 \implies uu1 = uu2 \implies eqExcPID2 (s (\!|news := uu1\!|))$
 $(s1 (\!|news := uu2\!|))$
 $\wedge uu1 uu2. eqExcPID2 s s1 \implies uu1 = uu2 \implies eqExcPID2 (s (\!|phase := uu1\!|))$
 $(s1 (\!|phase := uu2\!|))$

$\langle proof \rangle$

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 $title = title1 \wedge abstract = abstract1 \wedge content = content1 \wedge dis = dis1 \wedge decs = decs1$

$\langle proof \rangle$

lemma *cReview-step-eqExcPID2*:

assumes a :

$a = Cact (cReview cid uid p PID uid')$

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

shows $eqExcPID2 s s'$

$\langle proof \rangle$

9.2 Value Setup

type-synonym $value = userID \times userID\ set$

fun $\varphi :: (state,act,out)\ trans \Rightarrow bool$ **where**
 $\varphi (Trans - (Cact (cReview\ cid\ uid\ p\ pid\ uid'))\ ou\ -) =$
 $(pid = PID \wedge ou = outOK)$
 $|$
 $\varphi\ - = False$

fun $f :: (state,act,out)\ trans \Rightarrow value$ **where**
 $f (Trans\ s\ (Cact\ (cReview\ cid\ uid\ p\ pid\ uid'))\ -\ s') =$
 $(uid', \{uid'.\ isPC\ s\ cid\ uid' \wedge pref\ s\ uid'\ PID \neq Conflict\})$

lemma φ -def2:
 $\varphi (Trans\ s\ a\ ou\ s') =$
 $(ou = outOK \wedge$
 $(\exists\ cid\ uid\ p\ uid'.\ a = Cact\ (cReview\ cid\ uid\ p\ PID\ uid')))$
 $\langle proof \rangle$

fun $\chi :: act \Rightarrow bool$ **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 χ -def2:
 $\chi\ a =$
 $(\exists\ cid\ uid\ p\ n\ rc.\ a = Uact\ (uReview\ cid\ uid\ p\ PID\ n\ rc) \vee$
 $a = UUact\ (uuReview\ cid\ uid\ p\ PID\ n\ rc))$
 $\langle proof \rangle$

lemma $eqExcPID$ -step- φ -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 $\neg \varphi (Trans\ s1\ a\ ou1\ s1')$
 $\langle proof \rangle$

lemma $eqExcPID$ -step- φ :
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 $\varphi (Trans\ s\ a\ ou\ s^\wedge) = \varphi (Trans\ s1\ a\ ou1\ s1^\wedge)$
 ⟨proof⟩

lemma *non-eqExcPID-step- φ -imp*:
assumes $s: reach\ s$ **and** $ss1: eqExcPID\ s\ s1$
and $PID: PID \in \in paperIDs\ s\ cid$ **and** $ou: ou \neq outErr$
and $step: step\ s\ a = (ou, s^\wedge)$ **and** $step1: step\ s1\ a = (ou1, s1^\wedge)$
and $\varphi: \neg \varphi (Trans\ s\ a\ ou\ s^\wedge)$
shows $\neg \varphi (Trans\ s1\ a\ ou1\ s1^\wedge)$
 ⟨proof⟩

lemma *eqExcPID-step*:
assumes $s: reach\ s$ **and** $s1: reach\ s1$
and $ss1: eqExcPID\ s\ s1$
and $step: step\ s\ a = (ou, s^\wedge)$
and $step1: step\ s1\ a = (ou1, s1^\wedge)$
and $PID: PID \in \in paperIDs\ s\ cid$
and $ou-ph: ou \neq outErr \vee phase\ s\ cid > revPH$
and $\varphi: \neg \varphi (Trans\ s\ a\ ou\ s^\wedge)$ **and** $\chi: \neg \chi\ a$
shows $eqExcPID\ s'\ s1'$
 ⟨proof⟩

lemma *φ -step-eqExcPID2*:
assumes $\varphi: \varphi (Trans\ s\ a\ ou\ s^\wedge)$
and $s: step\ s\ a = (ou, s^\wedge)$
shows $eqExcPID2\ s\ s'$
 ⟨proof⟩

lemma *eqExcPID2-step*:
assumes $s: reach\ s$
and $ss1: eqExcPID2\ s\ s1$
and $step: step\ s\ a = (ou, s^\wedge)$
and $step1: step\ s1\ a = (ou1, s1^\wedge)$
and $PID: PID \in \in paperIDs\ s\ cid$ **and** $ph: phase\ s\ cid \geq revPH$
and $\varphi: \neg \varphi (Trans\ s\ a\ ou\ s^\wedge)$
shows $eqExcPID2\ s'\ s1'$
 ⟨proof⟩

lemma *eqExcPID2-step- φ -imp*:
assumes $s: reach\ s$ **and** $ss1: eqExcPID2\ s\ s1$
and $PID: PID \in \in paperIDs\ s\ cid$ **and** $ph: phase\ s\ cid > revPH$
and $step: step\ s\ a = (ou, s^\wedge)$ **and** $step1: step\ s1\ a = (ou1, s1^\wedge)$
and $\varphi: \neg \varphi (Trans\ s\ a\ ou\ s^\wedge)$
shows $\neg \varphi (Trans\ s1\ a\ ou1\ s1^\wedge)$
 ⟨proof⟩

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 $\varphi (Trans\ s\ a\ ou\ s^{\wedge}) = \varphi (Trans\ s1\ a\ ou1\ s1^{\wedge})$
<proof>

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 $\varphi: \neg \varphi (Trans\ s\ a\ ou\ s^{\wedge})$
shows $\neg \varphi (Trans\ s1\ a\ ou1\ s1^{\wedge})$
<proof>

end
theory *Reviewer-Assignment-NCPC*
imports *../Observation-Setup Reviewer-Assignment-Value-Setup Bounded-Deducibility-Security.Composition*
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 ⇒ bool* **where**
T (Trans - - ou s[^]) =
 $(\exists uid \in UIDs. \exists cid.$
 $PID \in \in paperIDs\ s'\ cid \wedge isPC\ s'\ cid\ uid \wedge pref\ s'\ uid\ PID \neq Conflict \wedge phase$
 $s'\ cid \geq revPH)$

term *isAUT*

declare *T.simps [simp del]*

definition $B :: \text{value list} \Rightarrow \text{value list} \Rightarrow \text{bool}$ **where**

$B \text{ vl vl1} \equiv$
 $\text{vl} \neq [] \wedge$
 $\text{length vl} = \text{length vl1} \wedge$
 $\text{distinct} (\text{map fst vl1}) \wedge \text{fst} ' (\text{set vl1}) \subseteq \text{snd} (\text{hd vl}) \wedge \text{snd} ' (\text{set vl1}) = \{\text{snd} (\text{hd vl})\}$

interpretation $BD\text{-Security}\text{-IO}$ **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** $B = B$
 $\langle \text{proof} \rangle$

lemma $\text{reachNT}\text{-non}\text{-isPC}\text{-isChair}$:

assumes $\text{reachNT } s$ **and** $\text{uid} \in \text{UIDs}$

shows

$(\text{PID} \in \text{paperIDs } s \text{ cid} \wedge \text{isPC } s \text{ cid uid} \longrightarrow$
 $\text{pref } s \text{ uid PID} = \text{Conflict} \vee \text{phase } s \text{ cid} < \text{revPH}) \wedge$
 $(\text{PID} \in \text{paperIDs } s \text{ cid} \wedge \text{isChair } s \text{ cid uid} \longrightarrow$
 $\text{pref } s \text{ uid PID} = \text{Conflict} \vee \text{phase } s \text{ cid} < \text{revPH})$
 $\langle \text{proof} \rangle$

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 $\neg \gamma (\text{Trans } s \ a \ \text{ou } s')$

$\langle \text{proof} \rangle$

lemma $T\text{-}\varphi\text{-}\gamma\text{-stronger}$:

assumes $s: \text{reach } s$ **and** $0: \text{PID} \in \text{paperIDs } s \ \text{cid}$

and $2: \text{step } s \ a = (\text{ou}, s') \varphi (\text{Trans } s \ a \ \text{ou } s')$

and $1: \forall \text{uid} \in \text{UIDs}. \text{isChair } s \ \text{cid} \ \text{uid} \longrightarrow \text{pref } s \ \text{uid} \ \text{PID} = \text{Conflict} \vee \text{phase } s \ \text{cid} < \text{revPH}$

shows $\neg \gamma (\text{Trans } s \ a \ \text{ou } s')$

$\langle \text{proof} \rangle$

lemma $T\text{-}\varphi\text{-}\gamma\text{-1}$:

assumes $s: \text{reachNT } s$ **and** $s1: \text{reach } s1$ **and** $\text{PID}: \text{PID} \in \text{paperIDs } s \ \text{cid}$

and $ss1: \text{eqExcPID } s \ s1$

and $\text{step1}: \text{step } s1 \ a = (\text{ou1}, s1') \ \text{and} \ \varphi1: \varphi (\text{Trans } s1 \ a \ \text{ou1 } s1')$

and $\varphi: \neg \varphi (\text{Trans } s \ a \ \text{ou } s')$

shows $\neg \gamma (\text{Trans } s1 \ a \ \text{ou1 } s1')$

$\langle \text{proof} \rangle$

lemma $\text{notIsPCorConflict}\text{-eqExcPID}\text{-roles}\text{-eq}$:

assumes $s: \text{reach } s$ **and** $s1: \text{reach } s1$ **and** $\text{PID}: \text{PID} \in \text{paperIDs } s \ \text{cid}$

and $pc: \neg \text{isPC } s \ \text{cid} \ \text{uid} \vee \text{pref } s \ \text{uid} \ \text{PID} = \text{Conflict}$

and $\text{eeq}: \text{eqExcPID } s \ s1$

shows $roles\ s\ cid\ uid = roles\ s1\ cid\ uid$
 ⟨proof⟩

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⟩

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⟩

lemma *eqExcPID-step-φ-eqExcPID-out*:
assumes $s: reach\ s$ **and** $s1: reach\ s1$
and $a: a = Cact\ (cReview\ cid\ uid\ p\ PID\ uid')$
and $a1: a1 = Cact\ (cReview\ cid\ uid\ p\ PID\ uid1')$
and $ss1: eqExcPID\ s\ s1$ **and** $step: step\ s\ a = (outOK, s')$
and $pc: isPC\ s\ cid\ uid1' \wedge pref\ s\ uid1'\ PID \neq Conflict$
and $rv1: \neg isRev\ s1\ cid\ uid1'\ PID$ **and** $step1: step\ s1\ a1 = (ou1, s1')$
shows $eqExcPID\ s'\ s1' \wedge ou1 = outOK$
 ⟨proof⟩

lemma *eqExcPID-ex-isNthReview*:
assumes $s: reach\ s$ **and** $s1: reach\ s1$ **and** $e: eqExcPID\ s\ s1$
and $i: isRevNth\ s\ cid\ uid\ PID\ n$
shows $\exists uid1. isRevNth\ s1\ cid\ uid1\ PID\ n$
 ⟨proof⟩

lemma *eqExcPID-step-χ1*:
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⟩

lemma *eqExcPID-step-χ2*:
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
 $\exists\ s1'\ uid1\ p.$
 $isRevNth\ s1\ cid\ uid1\ PID\ n \wedge$
 $step\ s1\ (UUact\ (uuReview\ cid\ uid1\ p\ PID\ n\ rc)) = (outOK, s1') \wedge$
 $eqExcPID\ s'\ s1'$
 $\langle proof \rangle$

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**
 $\Delta e\ s\ vl\ s1\ vl1 \equiv$
 $vl \neq \square \wedge$
 $($
 $(\exists\ cid.\ PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid \geq revPH \wedge$
 $\neg (\exists\ uid.\ isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict))$
 \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 $istate-\Delta1:$
assumes $B: B\ vl\ vl1$
shows $\Delta1\ istate\ vl\ istate\ vl1$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta 1$* : *unwind-cont* $\Delta 1$ $\{\Delta 1, \Delta 2, \Delta e\}$
 ⟨*proof*⟩

lemma *not- φ -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 $isRev\ s'\ cid\ uid\ PID = isRev\ s\ cid\ uid\ PID \wedge isPC\ s'\ cid\ uid = isPC\ s\ cid\ uid$

⟨*proof*⟩

lemma *γ -not χ -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*⟩

lemma *exists-failedAct*:

$\exists a. step\ s\ a = (outErr, s) \wedge userOfA\ a = uid$

⟨*proof*⟩

lemma *unwind-cont- $\Delta 2$* : *unwind-cont* $\Delta 2$ $\{\Delta 2, \Delta 3, \Delta e\}$

⟨*proof*⟩

lemma *unwind-cont- $\Delta 3$* : *unwind-cont* $\Delta 3$ $\{\Delta 3, \Delta e\}$

⟨*proof*⟩

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* ($K1exit\ cid$)

⟨*proof*⟩

lemma *noVal-K1exit*: *noVal* ($K1exit\ cid$) v

⟨*proof*⟩

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 $isPC\ s'\ cid\ uid' = isPC\ s\ cid\ uid'$
<proof>

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 $pref\ s'\ uid\ pid = pref\ s\ uid\ pid$

<proof>

lemma *invarNT-K2exit*: $invarNT\ (\lambda\ s.\ K2exit\ cid\ s\ v)$

<proof>

lemma *noVal-K2exit*: $noVal2\ (K2exit\ cid)\ v$

<proof>

definition *K3exit* **where**

$K3exit\ cid\ s \equiv PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > revPH$

lemma *invarNT-K3exit*: $invarNT\ (K3exit\ cid)$

<proof>

lemma *noVal-K3exit*: $noVal\ (K3exit\ cid)\ v$

<proof>

lemma *unwind-exit-Δe*: $unwind-exit\ \Delta e$

<proof>

theorem *secure*: $secure$

<proof>

end

theory *Reviewer-Assignment-NCPC-Aut*

imports *../Observation-Setup Reviewer-Assignment-Value-Setup Bounded-Deducibility-Security.Compositiona*

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.
      ( $\exists$  cid. PID  $\in$  paperIDs s' cid  $\wedge$  isPC s' cid uid  $\wedge$  pref s' uid PID  $\neq$  Conflict
       $\wedge$  phase s' cid  $\geq$  revPH)
       $\vee$ 
      ( $\exists$  cid. PID  $\in$  paperIDs s' cid  $\wedge$  isAut s' cid uid PID  $\wedge$  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  $\neq$  []  $\wedge$ 
    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  $B = B$ 
  <proof>
```

```
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  $\in$  paperIDs s cid  $\wedge$  isChair s cid uid  $\longrightarrow$  pref s uid PID = Conflict  $\vee$ 
  phase s cid < revPH)
   $\wedge$ 
  (PID  $\in$  paperIDs s cid  $\wedge$  isAut s cid uid PID  $\longrightarrow$ 
  phase s cid < notifPH)
  <proof>
```

```
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')
  <proof>
```

lemma *T-φ-γ-stronger*:

assumes *s*: reach *s* **and** *0*: $PID \in \in paperIDs\ s\ cid$

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>

lemma *T-φ-γ-1*:

assumes *s*: reachNT *s* **and** *s1*: reach *s1* **and** *PID*: $PID \in \in paperIDs\ s\ cid$

and *ss1*: eqExcPID2 *s* *s1*

and *step1*: step *s1* *a* = (*ou1*, *s1'*) **and** $\varphi1$: φ (*Trans* *s1* *a* *ou1* *s1'*)

and φ : $\neg \varphi$ (*Trans* *s* *a* *ou* *s'*)

shows $\neg \gamma$ (*Trans* *s1* *a* *ou1* *s1'*)

<proof>

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>

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*: $PID \in \in paperIDs\ s\ cid$ **and** *ph*: phase *s* *cid* $\geq revPH$

and φ : $\neg \varphi$ (*Trans* *s* *a* *ou* *s'*) **and** $\varphi1$: $\neg \varphi$ (*Trans* *s1* *a* *ou1* *s1'*)

and *UIDs*: userOfA *a* $\in UIDs$

shows *ou* = *ou1*

<proof>

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**

$\Delta2\ s\ vl\ s1\ vl1 \equiv$

$\exists cid\ uid.$

$PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid = revPH \wedge$

$isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict \wedge$

$eqExcPID2\ s\ s1 \wedge$

$distinct\ (map\ fst\ vl1) \wedge$

$fst\ ' (set\ vl1) \subseteq \{uid'. isPC\ s\ cid\ uid' \wedge pref\ s\ uid'\ PID \neq Conflict\} \wedge$

$fst\ ' (set\ vl1) \cap \{uid'. isRev\ s1\ cid\ uid'\ PID\} = \{\} \wedge$

$snd\ ' (set\ vl1) \subseteq \{\{uid'. isPC\ s\ cid\ uid' \wedge pref\ s\ uid'\ PID \neq Conflict\}\}$

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 $\Delta 1\ istate\ vl\ istate\ vl1$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta 1$* : *unwind-cont* $\Delta 1\ \{\Delta 1, \Delta 2, \Delta e\}$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta 2$* : *unwind-cont* $\Delta 2\ \{\Delta 2, \Delta 3, \Delta e\}$
 $\langle proof \rangle$

lemma *unwind-cont- $\Delta 3$* : *unwind-cont* $\Delta 3\ \{\Delta 3, \Delta e\}$
 $\langle proof \rangle$

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)
 $\langle proof \rangle$

lemma *noVal-K1exit*: *noVal* (*K1exit* cid) v
 $\langle proof \rangle$

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 $isPC\ s'\ cid\ uid' = isPC\ s\ cid\ uid'$

<proof>

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 $pref\ s'\ uid\ pid = pref\ s\ uid\ pid$

<proof>

lemma *invarNT-K2exit*: *invarNT* ($\lambda\ s.\ K2exit\ cid\ s\ v$)

<proof>

lemma *noVal-K2exit*: *noVal2* (*K2exit cid*) *v*

<proof>

definition *K3exit where*

$K3exit\ cid\ s \equiv PID \in \in paperIDs\ s\ cid \wedge phase\ s\ cid > revPH$

lemma *invarNT-K3exit*: *invarNT* (*K3exit cid*)

<proof>

lemma *noVal-K3exit*: *noVal* (*K3exit cid*) *v*

<proof>

lemma *unwind-exit-Δe*: *unwind-exit* Δe

<proof>

theorem *secure*: *secure*

<proof>

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 $trace\text{-}between :: state \Rightarrow (state, act, out) \text{ trans } trace \Rightarrow state \Rightarrow bool$
where

$empty[simp]: trace\text{-}between s [] s$
 $| step: \llbracket trace\text{-}between s \ tr \ sh; \ step \ sh \ a = (ou, s') \rrbracket \Longrightarrow trace\text{-}between s (tr@[Trans \ sh \ a \ ou \ s']) s'$

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* $s \ (tr1@tr2) \ s'$

$\longleftrightarrow (\exists sh. \text{trace-between } s \ tr1 \ sh \wedge \text{trace-between } sh \ tr2 \ s')$

<proof>

lemma *trace-ft-Cons*: *trace-between* $s \ (trn\#tr) \ s'$

$\longleftrightarrow (\exists sh \ ou \ a. \ trn = \text{Trans } s \ a \ ou \ sh \wedge \text{step } s \ a = (ou, sh) \wedge \text{trace-between } sh \ tr \ s')$

<proof>

lemmas *trace-ft-simps = trace-ft-empty trace-ft-snoc trace-ft-Cons trace-ft-append*

inductive *trace-to* :: $(state, act, out) \ \text{trans } \ \text{trace} \Rightarrow \ \text{state} \Rightarrow \ \text{bool}$ **where**

empty: *trace-to* $[] \ \text{istate}$

| *step*: $[[\text{trace-to } tr \ s; \ \text{step } s \ a = (ou, s')]] \Rightarrow \ \text{trace-to } (tr@[Trans \ s \ a \ ou \ s']) \ s'$

lemma *trace-to-ft*: *trace-to* $tr \ s \ \longleftrightarrow \ \text{trace-between } \text{istate} \ tr \ s$

<proof>

inductive-simps *trace-to-empty*[simp]: *trace-to* $[] \ s$

lemma *trace-to-reach*: **assumes** *trace-to* $tr \ s$ **shows** *reach* s

<proof>

lemma *reach-to-trace*: **assumes** *reach* s **obtains** tr **where** *trace-to* $tr \ s$

<proof>

lemma *reach-trace-to-conv*: *reach* $s \ \longleftrightarrow \ (\exists \ tr. \ \text{trace-to } tr \ s)$

<proof>

thm *trace-to.induct*[no-vars]

lemma *trace-to-induct*[case-names *empty step*, *induct set*]:

$[[\text{trace-to } x1 \ x2; \ P \ [] \ \text{istate};$

$\wedge \ tr \ s \ a \ ou \ s'. \ \text{step } s \ a = (ou, s')]]$

$\Rightarrow \ P \ (tr \ ## \ \text{Trans } s \ a \ ou \ s') \ s']]$

$\Rightarrow \ P \ x1 \ x2$

<proof>

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: $\llbracket \text{trn} = \text{Trans} - (\text{Cact} (\text{cPaper } \text{cid } \text{uid} - \text{pid} - -)) \text{outOK} - \rrbracket$
 $\implies \text{isAut}' (\text{tr}@[\text{trn}]) \text{cid } \text{uid } \text{pid}$

| *co-author*: \llbracket
 $\text{isAut}' \text{tr } \text{cid } \text{uid}' \text{pid};$
 $\text{trn} = \text{Trans} - (\text{Cact} (\text{cAuthor } \text{cid } \text{uid}' - \text{pid } \text{uid})) \text{outOK} - \rrbracket$
 $\implies \text{isAut}' (\text{tr}@[\text{trn}]) \text{cid } \text{uid } \text{pid}$

| *irrelevant*: $\text{isAut}' \text{tr } \text{cid } \text{uid}' \text{pid} \implies \text{isAut}' (\text{tr}@[-]) \text{cid } \text{uid}' \text{pid}$

lemma justify-author:

assumes *trace-to tr s*
assumes *isAut s cid uid pid*
shows *isAut' tr cid uid pid*
<proof>

lemma author-justify:

assumes *trace-to tr s*
assumes *isAut' tr cid uid pid*
shows *isAut s cid uid pid*
<proof>

theorem isAut-eq: *trace-to tr s \implies isAut s cid uid pid \longleftrightarrow isAut' tr cid uid pid*

<proof>

10.3 Becoming a Conference Chair

inductive isChair' :: (state,act,out) trans trace \Rightarrow confID \Rightarrow userID \Rightarrow bool where

creator: $\llbracket \text{trn} = \text{Trans} - (\text{Cact} (\text{cConf } \text{cid } \text{uid} - - -)) \text{outOK} - \rrbracket$
 $\implies \text{isChair}' (\text{tr}@[\text{trn}]) \text{cid } \text{uid}$

| *add-chair*: $\llbracket \text{isChair}' \text{tr } \text{cid } \text{uid}'; \text{trn} = \text{Trans} - (\text{Cact} (\text{cChair } \text{cid } \text{uid}' - \text{uid})) \text{outOK} - \rrbracket$

$\implies \text{isChair}' (\text{tr}@[\text{trn}]) \text{cid } \text{uid}$

| *irrelevant*: $\llbracket \text{isChair}' \text{tr } \text{cid } \text{uid} \rrbracket \implies \text{isChair}' (\text{tr}@[-]) \text{cid } \text{uid}$

lemma justify-chair:

assumes *trace-to tr s*
assumes *isChair s cid uid*
shows *isChair' tr cid uid*
<proof>

lemma chair-justify:

assumes *trace-to tr s*
assumes *isChair' tr cid uid*
shows *isChair s cid uid*
<proof>

theorem *isChair-eq*: $trace\text{-}to\ tr\ s \implies isChair\ s\ cid\ uid = isChair'\ tr\ cid\ uid$

<proof>

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*: $\llbracket isChair'\ tr\ cid\ uid'; trn = Trans - (Cact\ (cPC\ cid\ uid' - uid))\ outOK - \rrbracket$

$\implies isPC'\ (tr@[trn])\ cid\ uid$

| *irrelevant*: $\llbracket isPC'\ tr\ cid\ uid \rrbracket \implies isPC'\ (tr@[-])\ cid\ uid$

lemma *justify-com*:

assumes *trace-to tr s*

assumes *isPC s cid uid*

shows *isPC' tr cid uid*

<proof>

lemma *com-justify*:

assumes *trace-to tr s*

assumes *isPC' tr cid uid*

shows *isPC s cid uid*

<proof>

theorem *isPC-eq*: $trace\text{-}to\ tr\ s \implies isPC\ s\ cid\ uid = isPC'\ tr\ cid\ uid$

<proof>

10.5 Being a Reviewer

inductive *isRev'* :: $(state, act, out)\ trans\ trace \implies confID \implies userID \implies paperID \implies bool$ **where**

add-rev: $\llbracket isChair'\ tr\ cid\ uid'; trn = Trans - (Cact\ (cReview\ cid\ uid' - pid\ uid))\ outOK - \rrbracket$

$\implies isRev'\ (tr@[trn])\ cid\ uid\ pid$

| *irrelevant*: $\llbracket isRev'\ tr\ cid\ uid\ pid \rrbracket \implies isRev'\ (tr@[-])\ cid\ uid\ pid$

lemma *justify-rev*:

assumes *trace-to tr s*

assumes *isRev s cid uid pid*

shows *isRev' tr cid uid pid*

<proof>

lemma *rev-justify*:

assumes *trace-to tr s*

assumes *isRev' tr cid uid pid*

shows *isRev s cid uid pid*

<proof>

theorem *isRev-eq*: $trace\text{-}to\ tr\ s \implies isRev\ s\ cid\ uid\ pid = isRev'\ tr\ cid\ uid\ pid$

<proof>

10.6 Conflicts

fun *irrev-conflict* :: $userID \Rightarrow paperID \Rightarrow (state,act,out)\ trans \Rightarrow bool$

where

irrev-conflict uid pid (*Trans* - (*Cact* (*cPaper* - *uid'* - *pid'* - -)) *outOK* -)
 $\longleftrightarrow uid'=uid \wedge pid'=pid$
| *irrev-conflict uid pid* (*Trans* - (*Cact* (*cAuthor* - - - *pid'* *uid'*)) *outOK* -)
 $\longleftrightarrow uid'=uid \wedge pid'=pid$
| *irrev-conflict uid pid* - $\longleftrightarrow False$

fun *set-conflict* :: $userID \Rightarrow paperID \Rightarrow (state,act,out)\ trans \Rightarrow bool$

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 $\exists cid.\ isAut\ s\ cid\ uid\ pid$

<proof>

lemma *irrev-conflict-impl-conflict*:

assumes *trace-to tr s*

assumes $\exists trn \in set\ tr.\ irrev\ conflict\ uid\ pid\ trn$

shows $\text{pref } s \text{ uid } pid = \text{Conflict}$
 ⟨proof⟩

lemma *conflict-justify*:
assumes *TR*: *trace-to tr s*
assumes *conflict-trace uid pid tr*
shows $\text{pref } s \text{ uid } pid = \text{Conflict}$
 ⟨proof⟩

lemma *justify-conflict*:
assumes *TR*: *trace-to tr s*
assumes $\text{pref } s \text{ uid } pid = \text{Conflict}$
shows *conflict-trace uid pid tr*
 ⟨proof⟩

theorem *conflict-eq*:
assumes *trace-to tr s*
shows $\text{pref } s \text{ uid } pid = \text{Conflict} \longleftrightarrow \text{conflict-trace uid pid tr}$
 ⟨proof⟩

10.7 Conference Phases

fun *is-uPhase* **where**
 $\text{is-uPhase } cid \text{ (Trans - (Uact (uConfA } cid' \text{ - -)) outOK -)} \longleftrightarrow cid' = cid$
 $\text{is-uPhase } cid \text{ (Trans - (Uact (uPhase } cid' \text{ - -)) outOK -)} \longleftrightarrow cid' = cid$
 $\text{is-uPhase - -} \longleftrightarrow \text{False}$

inductive *phase'* :: $(\text{state}, \text{act}, \text{out}) \text{ trans trace} \Rightarrow \text{confID} \Rightarrow \text{nat} \Rightarrow \text{bool}$ **where**
initial: $\text{phase}' [] \text{ cid noPH}$
 $\text{approve: } \llbracket \text{phase}' \text{ tr } cid \text{ noPH; } \text{trn} = \text{Trans } s \text{ (Uact (uConfA } cid \text{ (voronkov } s) \text{ -)) outOK -} \rrbracket$
 $\implies \text{phase}' (\text{tr}@[\text{trn}]) \text{ cid setPH}$
 $\text{advance: } \llbracket \text{trn} = (\text{Trans - (Uact (uPhase } cid \text{ uid - ph)) outOK -}; \text{isChair}' \text{ tr } cid \text{ uid} \rrbracket$
 $\implies \text{phase}' (\text{tr}@[\text{trn}]) \text{ cid ph}$
 $\text{irrelevant: } \llbracket \text{phase}' \text{ tr } cid \text{ ph; } \neg \text{is-uPhase } cid \text{ trn} \rrbracket \implies \text{phase}' (\text{tr}@[\text{trn}]) \text{ cid ph}$

lemma *justify-phase*:
assumes *trace-to tr s*
assumes $\text{phase } s \text{ cid} = \text{ph}$
shows $\text{phase}' \text{ tr } cid \text{ ph}$
 ⟨proof⟩

lemma *phase-justify*:
assumes *trace-to tr s*
assumes $\text{phase}' \text{ tr } cid \text{ ph}$
shows $\text{phase } s \text{ cid} = \text{ph}$
 ⟨proof⟩

```

theorem phase-eq:
  assumes trace-to tr s
  shows phase s cid = ph  $\longleftrightarrow$  phase' tr cid ph
  <proof>

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. Bauerei, 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. Bauerei, 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.